

# AllFusion<sup>™</sup> Harvest Change Manager

## Tutorial

5.11



Computer Associates<sup>™</sup>

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

© 2002 Computer Associates International, Inc. (CA)

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

# Contents

## Chapter 1: Harvest Basics

What Is Harvest? .....	1-1
Adaptability .....	1-2
Concurrent Development Support .....	1-2
Parallel Development Support .....	1-2
Information Tracking .....	1-2
Tool Integration .....	1-2
Open Architecture .....	1-3
Complete Solution .....	1-3
Usability .....	1-3
Harvest Objects .....	1-3
Projects .....	1-4
Life Cycles: States and Processes .....	1-5
Packages and Package Groups .....	1-6
Forms .....	1-6
Users and User Groups .....	1-7
Repositories .....	1-7
Views .....	1-7
Items .....	1-8
Versions .....	1-8
Access Control .....	1-8
Harvest Architecture .....	1-9
Main Window Features .....	1-10
Taskbar .....	1-11
Workspace .....	1-11
List View .....	1-11
Output Log .....	1-11

## Chapter 2: Before You Begin

Setup Instructions .....	2-1
--------------------------	-----

---

## Chapter 3: Harvest Administration

Instructions for This Chapter .....	3-1
Background Information .....	3-2
Sample Software Development Process .....	3-2
Step 1: Log in to Harvest .....	3-3
Step 2: Create User Groups and Users .....	3-4
Creating User Groups .....	3-5
Creating Users .....	3-5
Step 3: Create a Project .....	3-8
Step 4: Create Working Views .....	3-9
Step 5: Create the Life Cycle States .....	3-10
Step 6: Create the Life Cycle Processes .....	3-12
Assign State .....	3-13
Coding State .....	3-15
Test State .....	3-22
Release State .....	3-27
Snapshot State .....	3-29
Step 7: Create a Repository .....	3-29
Step 8: Establish the Baseline .....	3-31
Step 9: Set the Access Permissions .....	3-33
Step 10: Review the Life Cycle .....	3-37
Step 11: Creating a Life Cycle Template .....	3-38
Let's Review .....	3-38

## Chapter 4: Using a Harvest Project

Instructions for This Chapter .....	4-1
Background Information .....	4-1
Step 1: Create a Package .....	4-2
Step 2: Promote the Package to Coding .....	4-4
Step 3: Check Out an Item For Update .....	4-5
Updating the File .....	4-7
Step 4: Check In the Change .....	4-8
Step 5: Approve the Package .....	4-10
Step 6: Promote the Package to Test .....	4-11
Step 7: Execute a Compare Views .....	4-12
Step 8: Perform a Visual Difference .....	4-13
Step 9: Remove an Item .....	4-15
Step 10: Delete a Version .....	4-16
Step 11: Demote the Package .....	4-16
In Review .....	4-18

---

## Chapter 5: Advanced Exercises

Instructions for This Chapter .....	5-1
Concurrent Development .....	5-1
Step 1: Create Two Packages .....	5-2
Step 2: Check Out the File for Concurrent Update .....	5-2
Step 3: Edit the File .....	5-5
Step 4: Check In the File .....	5-5
Step 5: Check Out for Concurrent Update Again .....	5-7
Step 6: Edit the File .....	5-8
Step 7: Check In the File .....	5-8
Review the Current Status .....	5-9
Step 8: Concurrent Merge .....	5-10
Step 9: Interactive Merge .....	5-12
Step 10: Approve the Packages .....	5-14
Step 11: Promote the Packages .....	5-14
Snapshots .....	5-15
Step 1: Create a Snapshot View .....	5-16
Step 2: Duplicate a Project .....	5-19
Step 3: Establish the Baseline .....	5-19
Step 4: Run the List Versions Process .....	5-20



# Harvest Basics

---

This chapter introduces important background information that enables you to better understand the hands-on exercises in later chapters. More detailed information can be found in the *Administrator Guide* and the *User Guide*.

## What Is Harvest?

Today's development teams build large, distributed application systems. They work from heterogeneous platforms at remote locations and make simultaneous changes to a multitude of interrelated software modules and system documentation. The only way to effectively track this complex, enterprise-wide development activity is with a comprehensive, repository-based change and configuration management (CCM) solution. Manual methods and simple file-control systems are not robust enough to help you improve delivery and bolster service levels.

Harvest helps you synchronize development activities on heterogeneous platforms, across your enterprise and throughout the application development life cycle. Harvest scales up to serve project teams working on your largest client/server enterprise systems and scales down to meet the needs of your smallest development teams.

Harvest is a client/server application. The client and server portions can both be executed on the same machine or be distributed across multiple platforms. The client portion of Harvest consists of the graphical user interface (GUI) and the command line interface. The server portion contains most of the program logic.

Following are some of the important benefits of Harvest:

- Readily adapts to your development process.
- Enables synchronization of concurrent development activities.
- Allows you to develop multiple projects in parallel, and then merge them together as needed.
- Allows you to track information associated with every change.
- Facilitates integration with other development and maintenance tools through an open architecture and application-programming interface.

- Employs an open architecture to enable easy access to CCM information.
- Provides a complete solution that encompasses all facets of the software development and maintenance process.
- Provides a consistent interface across all development and maintenance platforms in the environment.
- Is easy to use.

### Adaptability

Harvest helps you create and modify models of your own development life cycle and processes through simple point-and-click operations. It then uses this life cycle model to keep software changes under control, schedules on track, and everyone up to date. As changes are made to the development process, the model can easily be updated to support the changes.

### Concurrent Development Support

The concurrent development feature of Harvest allows more than one developer to work on the same area of code simultaneously without fear of overwriting each other's changes. Merge utilities are used to resolve any conflicts between the versions.

### Parallel Development Support

Harvest makes it possible to maintain multiple releases of the same application. For example, you could work on emergency fixes for a release while at the same time working on the next major version of that product. If necessary, changes made in the emergency area can be merged into the main product development cycle.

### Information Tracking

Harvest allows you to track pertinent information associated with every change: from the request that initiated the change, to data on who made the change and when, to the final resolution.

### Tool Integration

Interfaces from popular integrated development environments (IDEs), such as Visual Basic and Visual C++, allow developers to perform routine CCM functions without leaving the IDE.



## Open Architecture

Harvest's open architecture allows easy access to CCM information. Rather than developing yet another database standard, control information is stored within a commercially available relational database. Harvest table formats are fully documented. Database information is normally accessed from the GUI, but a site can access the database directly to generate reports or integrate with other development tools.

## Complete Solution

All functional groups involved in the development and maintenance process can benefit from Harvest, not just programmers. The entire development process can be controlled, including problem tracking, change management, builds, testing, quality assurance, documentation, and release. Extensive management reporting capabilities support auditing functions throughout the entire process.

## Usability

The Harvest user interface is implemented using state-of-the-art GUI technology. It has been designed with ease of use and minimal training time as primary criteria. Icons and toolbars are used throughout the interface, and keyboard accelerators are available for most common functions. Complete online help is also available.

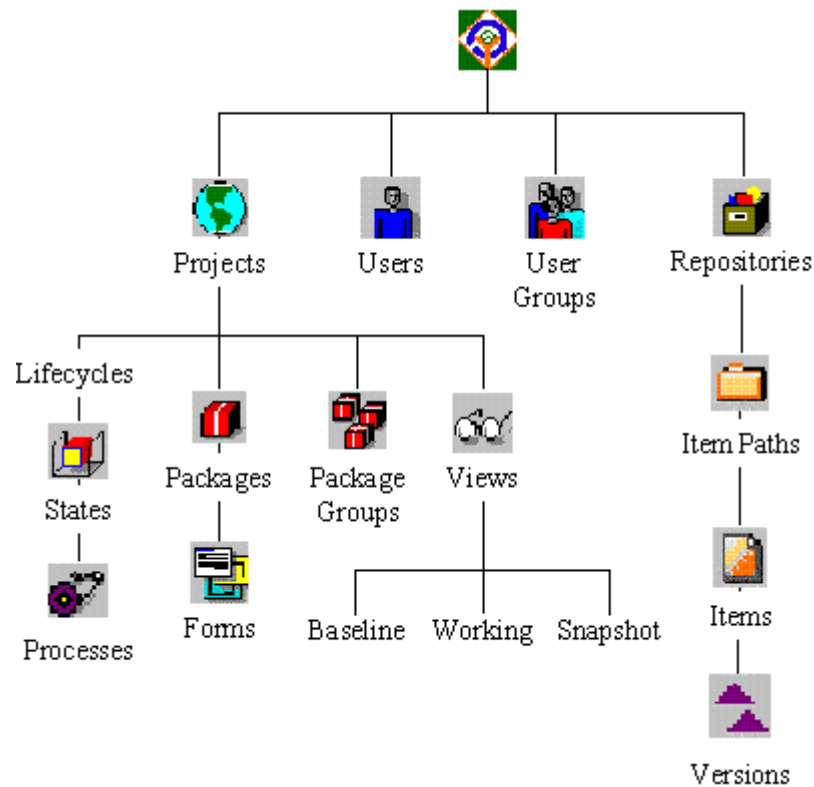
## Harvest Objects

Harvest incorporates an object-oriented user interface. Each icon represents a particular object type. Each object type represents different data and has specific actions that can be applied to it.

Complex CCM solutions can be built using a small number of basic Harvest object types. These objects include:

- Projects
- Life cycles
- Views
- Packages and package groups
- Forms

The following figure illustrates the hierarchy of the various objects in Harvest.



Each Harvest object is briefly introduced in the sections that follow. More in-depth discussions can be found in the *Administrator Guide* and the *User Guide*.

## Projects

The term *project* refers to the entire control framework within Harvest that supports a particular development or maintenance activity. A project includes information about what data to access, how changes progress through the cycle, the activities that can occur, and user responsibilities.

You can have many different projects, depending on the applications being controlled and the kind of development activity undertaken. For example, there might be one project for maintaining an already released version of an application, another for developing the next release and a third for maintaining code shared among various applications. There might be an entirely different project used by the support group for tracking incidents and problems.

## Life Cycles: States and Processes

A *life cycle* describes the path that changes take as development progresses in a project in terms of an ordered set of phases or states. The life cycle can be considered the "heart" of a project because it controls the flow of development life within it.

A life cycle can include many or few states, depending on each site's individual requirements. A common development scenario might include states for assigning change requests, making changes, testing changes, integrating changes, and releasing a completed product.

A set of valid *processes* can be defined for each state in the life cycle. Processes are commands that perform a task. The processes defined for a state determine the activities that can be performed, or its scope of work.

Additional actions can be associated with each process, based on the success of its execution. This allows commands to be linked together to perform more complex tasks. A user must explicitly invoke processes associated with a state. Linked processes occur automatically whenever the process to which they are linked is invoked.

Harvest comes with many process types that can be configured by an administrator. In addition, user-defined processes (UDPs) allow you to define additional processes to Harvest and enable you to easily integrate other tools or utilities into the CCM environment.

The following table displays a subset of the processes supplied with Harvest.

Process Type	Action
Approve	Allows electronic sign-off of a package or package group before it can be promoted.
Check In	Creates a new version of an item by bringing the changes made to a file into Harvest.
Check Out	Copies a version of a Harvest item to an external directory and optionally reserves it.
Compare Views	A summary report that lists the differences between any two views.
Concurrent Merge	The first step of merging branch versions. Compares a selected branch version with the latest trunk version to determine conflicts.
Create Package	Creates a package and, optionally, a form with the same name and automatically associates them.
Cross Project Merge	Merges versions between Harvest projects.

Process Type	Action
Delete Version	Deletes the latest version(s) of an item.
Demote	Returns a package and its associated versions to a previous state.
Interactive Merge	Allows users to resolve conflicts between branch and trunk versions.
List Versions	A report that lists information about the versions of items in the Harvest repository.
Move Package	Moves a package from a state in one project to a state in another project.
Notify	Sends an email message to selected users or user groups.
Promote	Moves a package and its associated versions to another state further in the life cycle.
Remove Item	Logically deletes an item from the repository.
Take Snapshot	Takes a read-only image of a working view at a certain point in time.
User-Defined Process (UDP)	Executes a user-defined process.

## Packages and Package Groups

A *package* is the basic unit of work that moves through a life cycle. It typically represents a problem or a request that needs to be tracked, the changes made in response to the problem or incident and any other associated information. The package is the user's link to the actual data accessed during the change process. Each package resides within a particular state of the life cycle.

A *package group* provides a way to operate on related packages as a unit. A particular package might belong to one group or several, or it might not belong to any. Any operation that can be performed on a package can be performed on all packages in a package group. For example, a user can promote all packages in a group from one state to the next. If an approval is required before packages can be promoted, another user can approve the promotion of the group.

## Forms

Harvest *forms* can be used in much the same manner that paper forms are used. For example, they can be used to track issues and problems or as a structured method of communication.

Forms are always associated with packages. This association provides the link between the problem tracking and change management functions in Harvest. Forms can also be associated with other forms, allowing information to be cross-referenced. For example, you might associate a customer contact form with a problem reported by that customer. All the information in a form is directly accessible from the associated package.

A number of standard forms are provided with Harvest, such as a problem tracking form, user contact form, testing information form, question and answer form, and comment form. In addition, custom forms can be created that are tailored to the specific requirements of individual users.

## Users and User Groups

When *users* are defined in Harvest, they can be assigned to *user groups*. Users and user groups exist at the global level within Harvest, so they are available to all projects defined within a Harvest installation. A user can belong to any number of user groups, and there is no hierarchy implied by the groups. For example, a user in the Development Manager group does not implicitly have the privileges of users in the Developer group. User groups provide a powerful, flexible mechanism that can be employed in association with access control, notification, and approvals.

## Repositories

A *repository* is where the data under Harvest's control is stored. Repositories reside within the relational database. Typically, a repository is comprised of all files that make up an application, but applications can also be split up between multiple repositories. For example, you might store the code in one repository, the documentation in another, and the installation utilities in another. Repositories can also be shared by more than one project.

Repository data can either be accessed directly by a user with Harvest administrator access or other users can access it through views.

## Views

*Views* define what repository data can be accessed from within a project. Each project has one *baseline*, which defines the repository items available to the project. A baseline can include items from more than one repository, as in the case of shared code.

Additional views, called *working views*, can be created based on the baseline. Within a life cycle, each state can have a particular view associated with it. The view determines which item versions are accessible to users operating in a state using that view.

The third type of view, called a *snapshot view*, allows users to capture their application inventory at a specific point in time. This facilitates the re-creation of specific application versions.

## Items

Repositories are composed of *items*. An item in the repository is equivalent to a file on the file system. If you check out an item named `foo.c` from the repository, a file named `foo.c` will be created on your file system.

## Versions

Each item in a repository consists of *versions* of the item. Item versions are visible to users only through views; they cannot be seen by looking directly at the repository. *Base versions*, which are the initial versions of the items in a repository, are shown through both the baseline and all working views. Subsequent versions created after the baseline has been established are shown through working views only.

When a user checks in a new version of an item, that new version becomes visible only from within the states that use the same view as the state from which the user checked it in. For example, if a user checks in version 2 of `file1.c` from the Coding state, and the Unit Test state shares the same view as the Coding state, version 2 of `file1.c` will be visible in both states. However, version 2 will not be visible in the Release state because that state does not share the same view. In order for version 2 to be visible in the Release state, the package that was used to create the version must be promoted to the Release state.

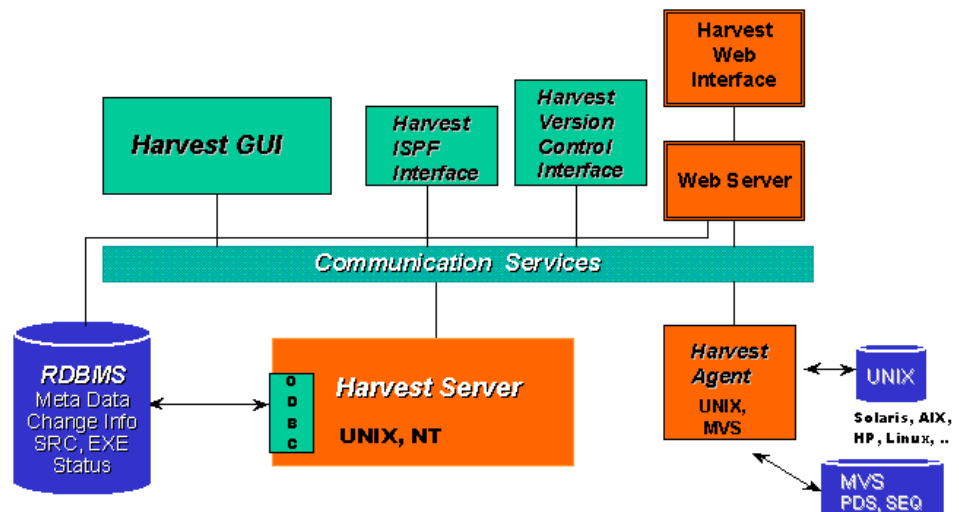
## Access Control

Harvest access control is designed to allow the implementation of any level of access control required. Access control is based on the combination of user or user group, object, and action. For example, by using Harvest's access methods, the Developer user group can be given access to check out an item from the Coding state.

## Harvest Architecture

Harvest is a client/server application built to support distributed development and take advantage of the strengths of various platforms. The client/server model used by Harvest is called the *application server* model. In this model, the client process presents data and manages keyboard and device input. The application logic is defined and processed remotely by a group of dedicated application servers. The application servers, in turn, provide access to database servers via standard SQL calls.

The following figure illustrates the various components in the Harvest architecture:



The Harvest *client* for PC platforms consists of the graphical user interface (GUI) and command line interface. MVS functions can be executed from the ISPF client interface or the REXX command line interface. The client is the program that handles the user interface. It receives input from the user and sends it to the server for processing. It also presents the returned information to the user.

The *server* program executes operations requested by the Harvest clients. It contains the Harvest business logic.

The *ODBC layer* enables communication between the server and a database. The database is used to store all information under Harvest's control.

A *communication layer* provides the connection between the Harvest client and server. The communication layer consists of a set of protocols that allow the Harvest components to work together.

Part of the communication layer is a program called the *broker*. Because there are many clients and servers in one network, the broker assigns each client an appropriate server. Each server registers with the broker and provides the broker with enough information so that the client can find the appropriate server when requesting service. Each client then requests the broker to connect to an available server.

For users to successfully invoke the Harvest client, the following conditions must be met:

- The relational database must be running.
- The Harvest broker must be running.
- At least one Harvest server process must be running.

Starting and stopping an Oracle database is discussed in the *Oracle RDBMS Runtime Getting Started Guide*. Information on managing the broker and server is in the *AllFusion Harvest Change Manager Administrator Guide*.

## Main Window Features

After logging in to Harvest, the main window is displayed. The main window is divided into four panes: the *taskbar* on the left, the *workspace* in the middle, the *list view* on the right, and the *output log* across the bottom.





## Taskbar

The taskbar is the left pane on the main window. If you have favorite or frequently used tasks (scripts or reports) to run, you can create a shortcut icon for each one on the taskbar. The taskbar can be divided into groups of shortcuts to help organize your information. You can add and remove groups, shortcuts, change the icons, and hide the taskbar.

## Workspace

The workspace shows all Harvest objects to which you have access. If you log into the Harvest Administrator window, the workspace will consist of five tab pages: Life cycles, Repositories, User Groups, Forms, and Files. If you log into the Harvest workbench, you will see only the Files and Projects tabs. The objects within each tab, with the exception of Forms, are displayed in a hierarchical format.

## List View

The list view displays information related to an object that is selected in the workspace. For example, if you select a user group in the workspace, the users that belong to that group will be displayed in the list view.

## Output Log

Most activities in Harvest generate output. This output can be the focus of an activity, such as in generating reports; or it is informational, documenting the results of an action. Output is displayed in the log at the bottom of the main window. Information in the output log can be copied or saved to an external file.



# Before You Begin

---

Before you can begin the tutorial, a few things need to be set up. Depending on your knowledge and responsibilities, you might be able to perform some or all of these tasks yourself; or someone, such as a system administrator, might need to complete some of them.

## Setup Instructions

1. Install Harvest.

Make sure Harvest is installed on both the server and the client. Instructions for installing Harvest are in the *Install Guide*.

2. Start a broker and server.

Make sure the Harvest broker process is running and that there is a server process available. For information on starting these processes, see the *Administrator Guide*.

3. Define a tutorial user.

Define a user name in Harvest that can be used by anyone wanting to use the tutorial. For example, you might use the name “Student” or “Tutorial.” This user needs to have Harvest Admin access. Write the name down, you will need this information later.

4. Create a working directory.

Create a working directory on your local machine; it can be any name and in any location, as long as you remember it. Some of the exercises require this directory to exist so files can be checked out to it. For example, you might create the directory D:\Tutorial Directory. Write the location down, you will need this information later.



# Harvest Administration

---

Before anyone can use Harvest to do change and configuration management, some administrative tasks must be performed. In this chapter, you will learn how to use all of the administrative features of Harvest. The administrative tasks you will perform include:

- Creating users and user groups
- Creating a project
- Creating views
- Creating a life cycle
- Setting up a repository
- Assigning access permissions

In general, the order in which you perform the activities in this chapter does not matter, but the steps are ordered in such a way as to make it easier to follow and quicker to set up.

## Instructions for This Chapter

You should follow the exercises in this chapter while using your client machine. Harvest must already be installed on both the server machine and your client machine. If it is not, refer to the *Install Guide* before continuing.

You will need to know the user name and password of the tutorial account that was set up in the previous chapter. You will also need to know the name of the machine on which the Harvest broker process is running.

## Background Information

The use of Harvest is divided into administrative tasks and day-to-day tasks. The administrative tasks are performed primarily when first implementing Harvest, and are typically performed by someone designated as a Harvest administrator. Such tasks include setting up the repository, defining the life cycle to be used, setting up user groups, and setting access permissions.

After a Harvest project is set up, day-to-day operation is straightforward. Users are given access to information, provided a set of processes that can be performed, and notified of actions that have been or need to be taken.

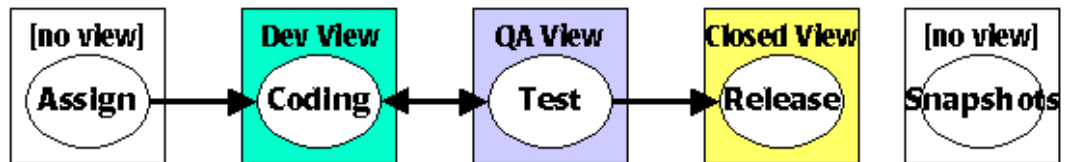
## Sample Software Development Process

You will be setting up a project that supports a typical, though simplified, software development process. The team consists of two developers, one quality assurance (QA) engineer, and one development manager. The software application being worked on is already on the market, so the purpose of this project is to support maintenance work on the current release of that software.

There are five phases in the development process:

- **Assign** - In this phase (state), modification requests (packages) are created by various personnel such as QA engineers, customer support personnel, and the development manager. The development manager is responsible for assigning a priority to each request and then passing the request to the development team (promoting it to the Coding state).
- **Coding** - In this state, the developers determine what code must be modified. They check out the code to their respective client machines, make the changes, and then check it in. When all code affected by the request is modified, the package is then approved by the development manager and promoted to the Test state.
- **Test** - In this state, the tester reviews the request and then tests the code to ensure it works according to the request. If errors are found in the code, the package is then given back to the developer (demoted to the Coding state). If no errors are found, the request package is promoted to the Release state.
- **Release** - This state is where all code that is ready to be released resides. Packages should never be demoted from this state. When the code is ready to be released, a kind of picture, called a snapshot, is taken of the code that will be included in that release.
- **Snapshots** - This state is where all snapshots reside. Snapshots make it easy to go back and look at the content of any release.

The following figure illustrates the basic layout of the software development process outlined above:



Notice that each state, except for Assign, has an associated view. Views are discussed later in this chapter under Step 4: Create Working Views and Step 8: Establish the Baseline.

## Step 1: Log in to Harvest

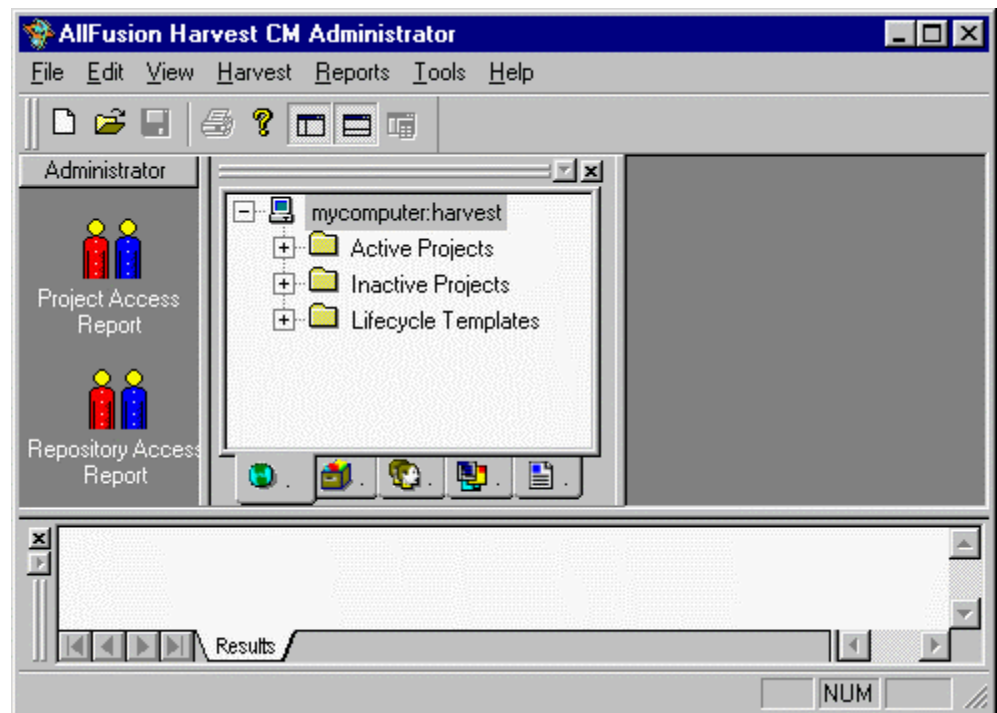
1. Double-click the Harvest program icon, or select Harvest Administrator from the Harvest program group under the Start menu.

The following login dialog will appear:



2. Enter the User Name and Password that you were given by the administrator. If you do not have this information, see the previous chapter.
3. In the Broker field, enter the name of the machine where the Harvest broker is running. This is usually, but not always, the Harvest server machine.
4. Enabling the Save Password check box will cause the password you entered to be retained for your next Harvest session.
5. Click OK.

Once you have successfully logged in, you will see the Administrator window:



## Step 2: Create User Groups and Users

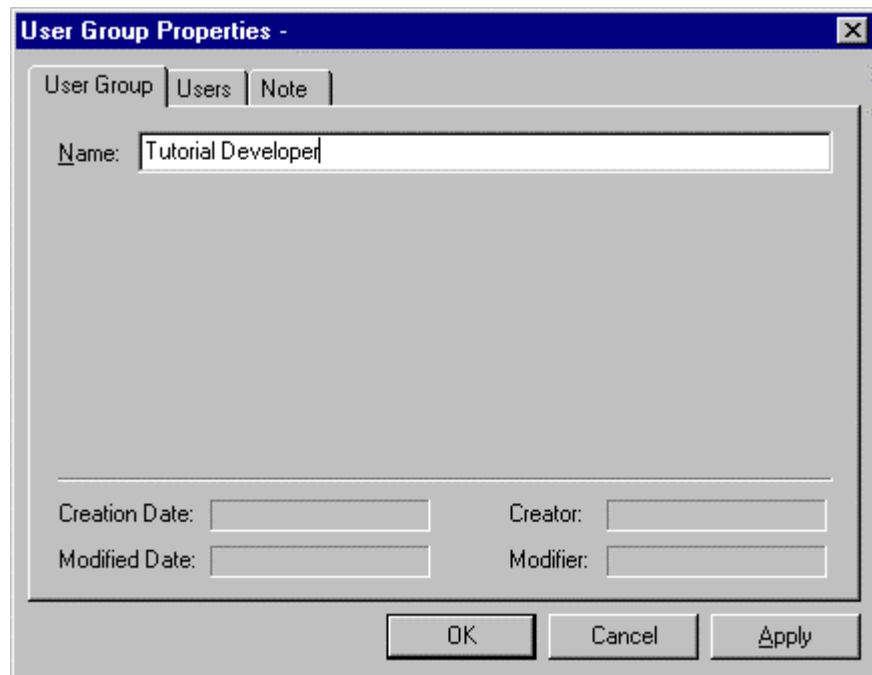
During this tutorial, you will be performing different tasks posing as different users. For example, when you create a package, you will be logged in as the user JimE.

Access permissions are set on the user group level, so each user must belong to a specific user group. First, you will create the appropriate user groups. Then, within each user group, you will create Harvest user accounts for each person you will be impersonating.



## Creating User Groups

1. Select the User Groups tab at the bottom of the workspace. You will see two folders: Users and User Groups.
2. Right-click the User Groups folder and choose New User Group from the shortcut menu. This will display the User Group Properties dialog.



3. Enter **Tutorial Developer** in the Name field, and then click OK.
4. Repeat steps 2 and 3, changing the name to **Tutorial QA**, and then click OK.
5. Repeat steps 2 and 3 again, changing the name to **Tutorial Dev. Manager**, and then click OK.

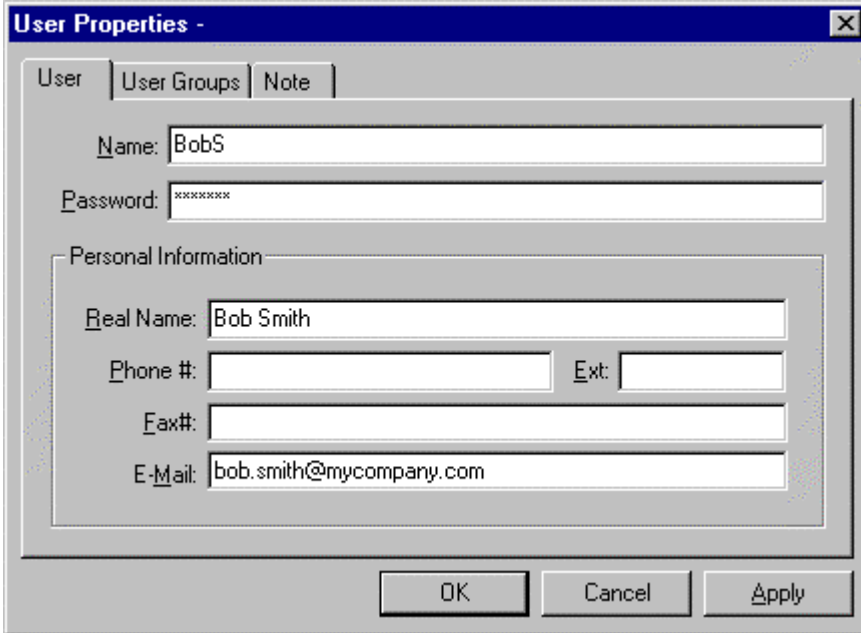
## Creating Users

In this section, you will create Harvest users and assign them to the appropriate user groups. First, you will create a user named BobS and add him to the Tutorial Developer group.

1. From the User Groups tab, select the Users folder.
2. Right-click and choose New User from the shortcut menu. The User Properties dialog will appear.
3. Enter **BobS** in the Name field. This will be the login name for Bob Smith and identifies him throughout Harvest.
4. Enter **harvest** in the Password field.

5. Enter **Bob Smith** in the Real Name field.
6. Enter your email address in the E-Mail field.
7. You do not need to fill in the Phone #, Ext, or Fax # fields.

The User Properties dialog should now look similar to the following:

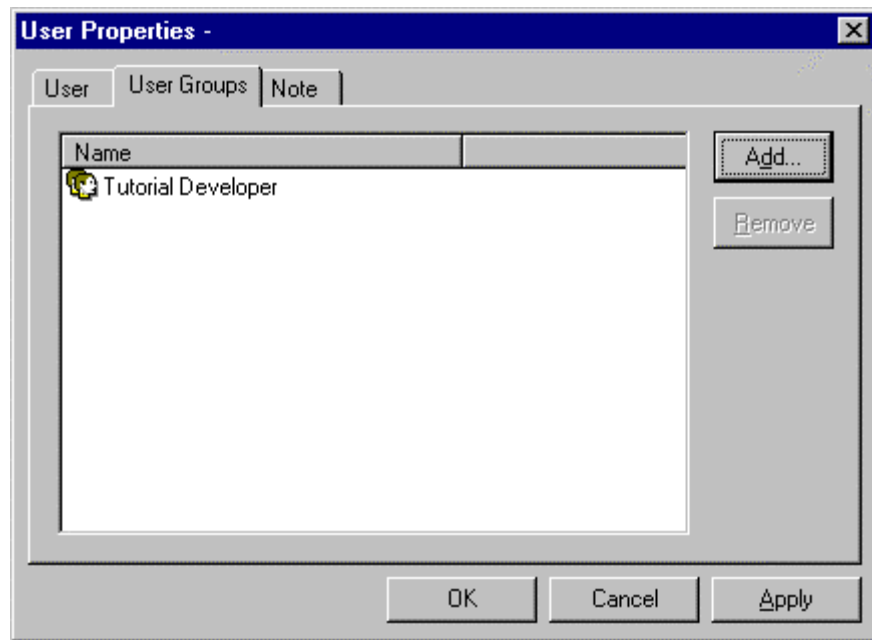


The image shows a Windows-style dialog box titled "User Properties -". It has three tabs: "User", "User Groups", and "Note". The "User" tab is selected. Inside the dialog, there are several text input fields. At the top, the "Name:" field contains "BobS". Below it, the "Password:" field contains a series of asterisks. A section titled "Personal Information" is enclosed in a rounded rectangle. Inside this section, the "Real Name:" field contains "Bob Smith". Below that, there are three fields: "Phone #:", "Ext:", and "Fax#:", all of which are empty. The "E-Mail:" field contains "bob.smith@mycompany.com". At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Apply".

You want to make Bob a member of the Tutorial Developer user group. To do this:

1. Click the User Groups tab on the User Properties dialog.
2. Click the Add button. A list of available user groups is displayed.

3. Select Tutorial Developer, and then click OK.



4. Click OK to close the User Properties dialog.

You will create a user named MaryT and add her to the Tutorial Developer group.

1. Right-click the Users folder and choose New User from the shortcut menu.
2. Enter **MaryT** in the Name field.
3. Enter **harvest** in the Password field.
4. Enter **Mary Turner** in the Real Name field.
5. Enter your email address in the E-Mail field.
6. You do not need to fill in the Phone #, Ext, or Fax # fields.
7. Click the User Groups tab and then click Add to display a list of available user groups.
8. Select Tutorial Developer and then click OK.
9. Click OK to close the User Properties dialog.

You will create a user named RickC and add him to the Tutorial QA group.

1. Right-click the Users folder and choose New User from the shortcut menu.
2. Enter **RickC** in the Name field.
3. Enter **harvest** in the Password field.
4. Enter **Rick Curtis** in the Real Name field.

5. Enter your email address in the E-Mail field.
6. You do not need to fill in the Phone #, Ext, or Fax # fields.
7. Click the User Groups tab and then click Add.
8. Select Tutorial QA and then click OK.
9. Click OK to close the User Properties dialog.

Finally, you will create a user named JimE and add him to the Tutorial Dev. Manager group.

1. Enter **JimE** in the Name field.
2. Enter **harvest** in the Password field.
3. Enter **Jim Evans** in the Real Name field.
4. Enter your email address in the E-Mail field.
5. You do not need to fill in the Phone #, Ext, or Fax # fields.
6. Click the User Groups tab and then click Add.
7. Select Tutorial Dev. Manager and then click OK.
8. Click OK to close the User Properties dialog.

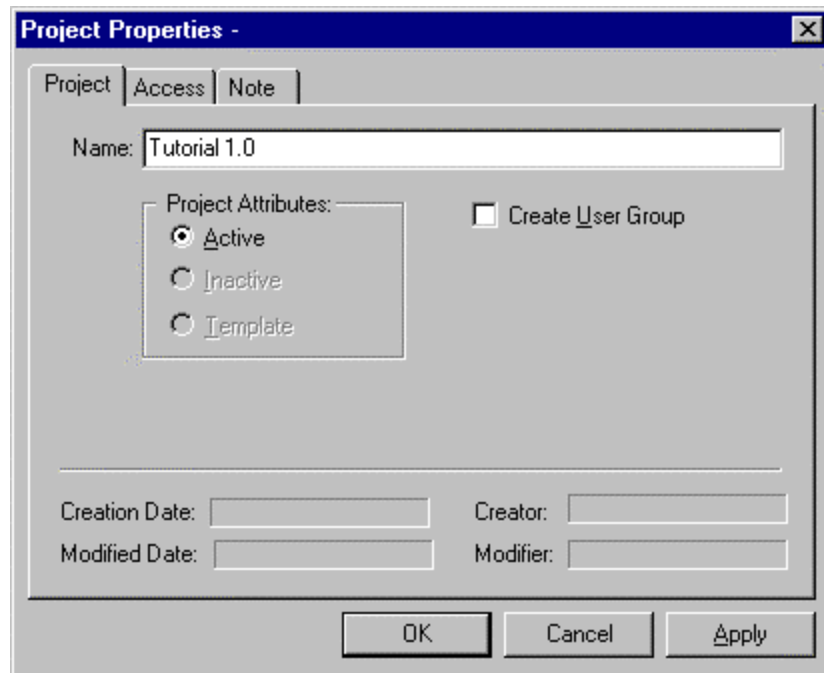
You are finished setting up your users and user groups. A summary of the users you created and the groups you assigned them to is listed in the following table:

User	User Group
BobS	Tutorial Developer
MaryT	Tutorial Developer
RickC	Tutorial QA
JimE	Tutorial Dev. Manager

## Step 3: Create a Project

You are ready to create a project.

1. Click the Lifecycle tab at the bottom of the workspace and then select the Active Projects folder.
2. Right-click and choose New Project from the shortcut menu. This will display the Project Properties dialog.
3. Enter **Tutorial 1.0** in the Name field.
4. Ensure that the Active radio button is selected. This activates the project so users can use it.



5. Click OK.

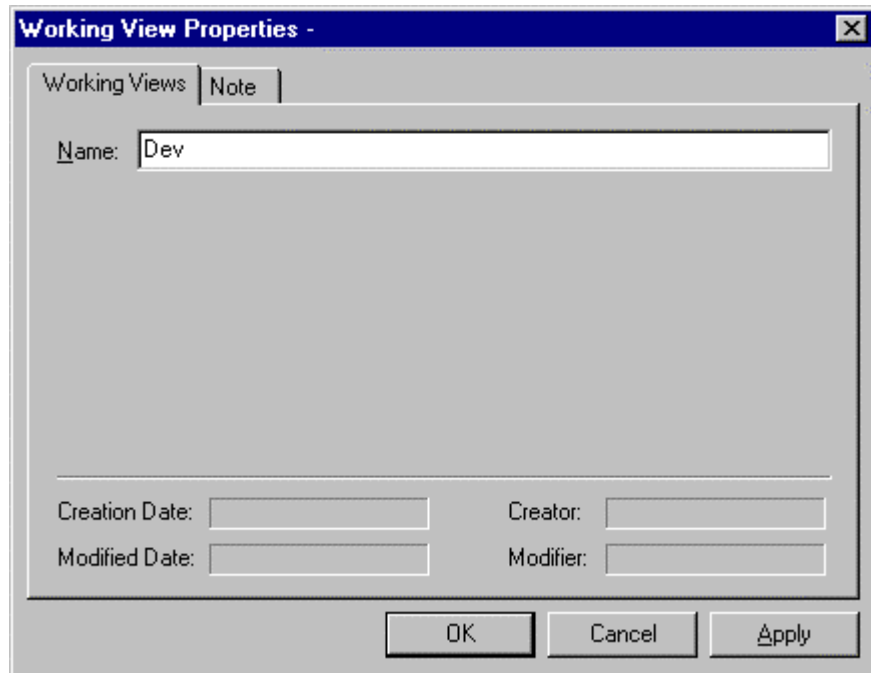
**Note:** When you create a project, a user group with the same name is automatically created if you enable the Create User Group option.

## Step 4: Create Working Views

A working view is like a window through which you can see the data under Harvest's control. When you check in a new version of an item, that version becomes visible within your current working view, but people using other views cannot see it. If you then promote the package, the new version is associated with a different state and view, the version then becomes visible to users using that view. Views will be discussed when you start to create states later in this chapter.

1. Click the plus (+) sign next to the Tutorial 1.0 project to expand it. You should see two folders: States and Data Views.
2. Click the plus (+) sign next to Data Views to expand it. You should see: Baseline, Working Views, and Snapshot Views.
3. Select Working Views.
4. Right-click and choose New Working View from the shortcut menu. The Working View Properties dialog will be displayed.

5. Enter **Dev** in the Name field.



6. Click OK.
7. Repeat steps 4, 5, and 6, changing the name of the view to **QA**.
8. Repeat steps 4, 5, and 6, changing the name of the view to **Closed**.

## Step 5: Create the Life Cycle States

As discussed earlier, our life cycle has been designed with five phases, or states. In this section, you will be creating those states and defining which view each one will use. In addition, you will be adding a state in which you can view all of the snapshots that are available.

1. If the Tutorial 1.0 project is not already expanded, click the plus (+) sign next to the Tutorial 1.0 project to expand it, and then select the States folder.
2. Right-click the States folder and choose New State from the shortcut menu. This will open the State Properties dialog.
3. In the Name field, enter **Assign**.

4. Because you do not need to be able to see or update any items in the repository from within the Assign state, you do not need to assign a view to the state. Accept the default, No View, from the View drop-down list.

The image shows a 'State Properties' dialog box with three tabs: 'State', 'Access', and 'Note'. The 'State' tab is selected. Inside the dialog, there are three main fields: 'Name' with the value 'Assign', 'View' with a dropdown menu showing 'No View' and an 'All snapshot' checkbox, and 'P.M. Status' with a dropdown menu showing 'None\_Defined'. At the bottom, there are four input fields for 'Creation Date', 'Creator', 'Modified Date', and 'Modifier'. At the very bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Apply'.

5. Click OK.
6. Repeat step 2 and enter the name of the new state, **Coding**.
7. Choose Dev from the View drop-down list, and then click OK.
8. Repeat step 2 and enter the name of the new state, **Test**.
9. Choose QA from the View drop-down list, and then click OK.
10. Repeat step 2 and enter the name of the new state, **Release**.
11. Choose Closed from the View drop-down list, and then click OK.
12. Repeat step 2 and enter the name of the new state, **Snapshots**.
13. Click the All Snapshot check box to enable it, and then click OK.

An alternative method of creating states is by right-clicking on a state and choosing Copy from the shortcut menu. You can then position your mouse in the project you want the state to be located, right-click and choose Paste from the shortcut menu.

## Step 6: Create the Life Cycle Processes

In this section, you will define the processes that can be executed from within each state. The table below lists each process you will create, and the name and options you will use for each process.

**Note:** If the option you want is not shown on the initial page of the properties dialog, click the Defaults tab to show additional options.

State	Process Type	Name	Options Enabled
Assign	Create Package	Create Package	Initial State=Assign Create Automatically=Problem Report Default Name=TMR#
	Promote	Promote to Coding	Promote To=Coding
Coding	Approve	Approve Package	Approval User=JimE
	Checkin	Check In Changes	Item Filter=New or existing items, Existing items only Mode=Update and release Directory Options=Preserve directory structure Delete Files After Check In=not selected
	Checkout	Check Out	Mode=Update, Concurrent Update Options=Preserve View Path Structure Replace Read Only Files=not selected
	Concurrent Merge	Concurrent Merge	Merge Options=all selected
	Interactive Merge	Interactive Merge	Option=3-Way Merge
	List Version	List Version	
	Promote	Promote to Test	Promote To=Test

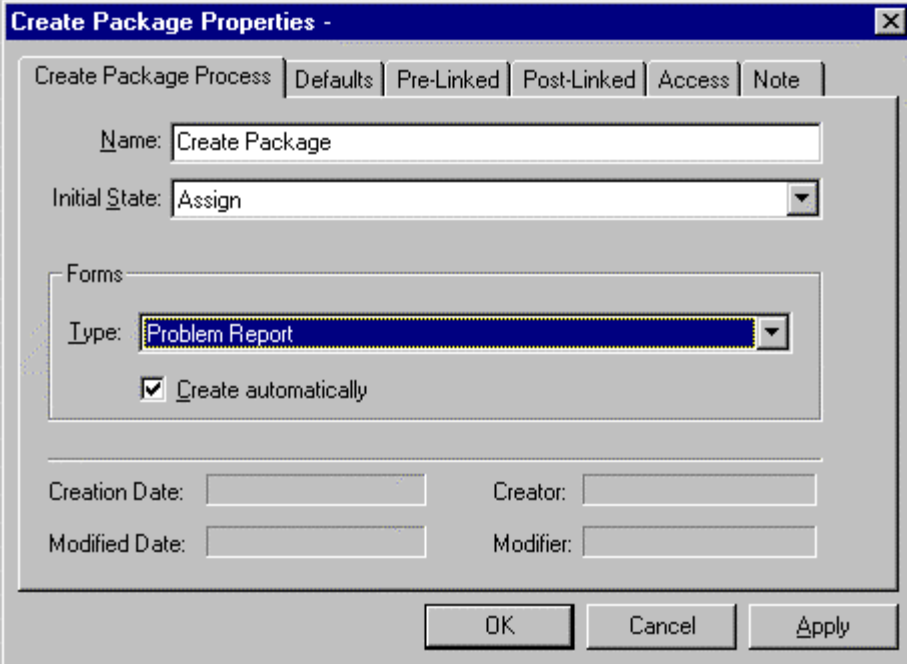


State	Process Type	Name	Options Enabled
Test	Checkout	Check Out	Mode=Browse Directory Options=Preserve View Path Structure Replace Read Only Files=not selected
	Compare Views	Compare Views	
	Delete Version	Delete Version	
	Demote	Demote to Coding	Demote To=Coding
	Promote	Promote to Release	Promote To=Release
	Remove Item	Remove Item	
Release	Take Snapshot view	Take Snapshot	

## Assign State

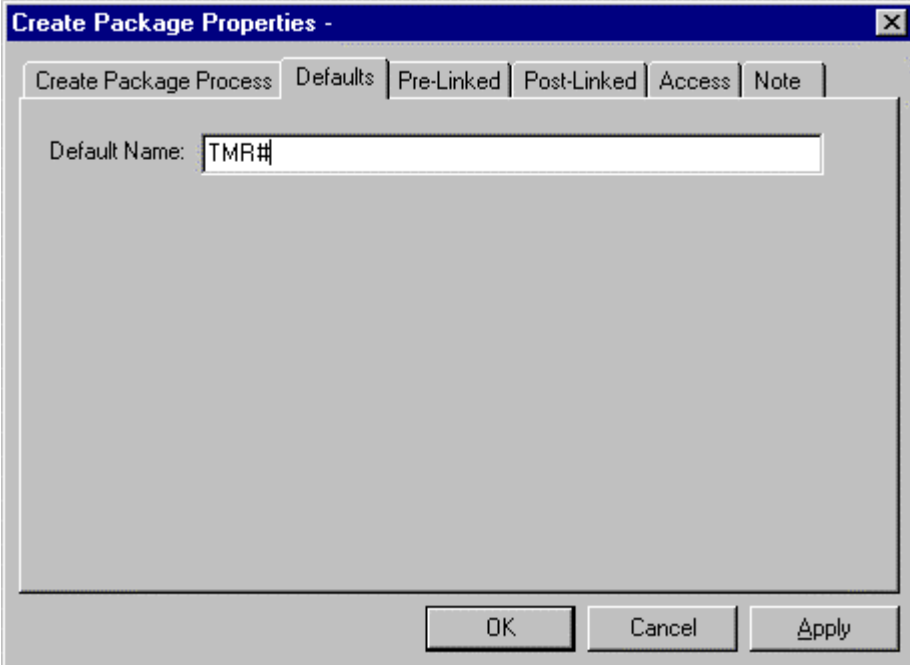
1. To show the states in the States folder, click the plus (+) sign next to the States folder to expand it.
2. Click the plus (+) sign next to the Assign state to expand it, and then select the Processes folder.
3. Right-click and choose New, Create Package Process from the shortcut menu. This will add a Create Package process to the Assign state.
4. Enter **Create Package** in the Name field.
5. Choose Assign from the Initial State drop-down list. This will ensure that new packages will be placed in the Assign state.
6. Click the Create automatically check box to enable it.
7. Choose Problem Report from the Type drop-down list.

When you are finished, the dialog should look like the following:



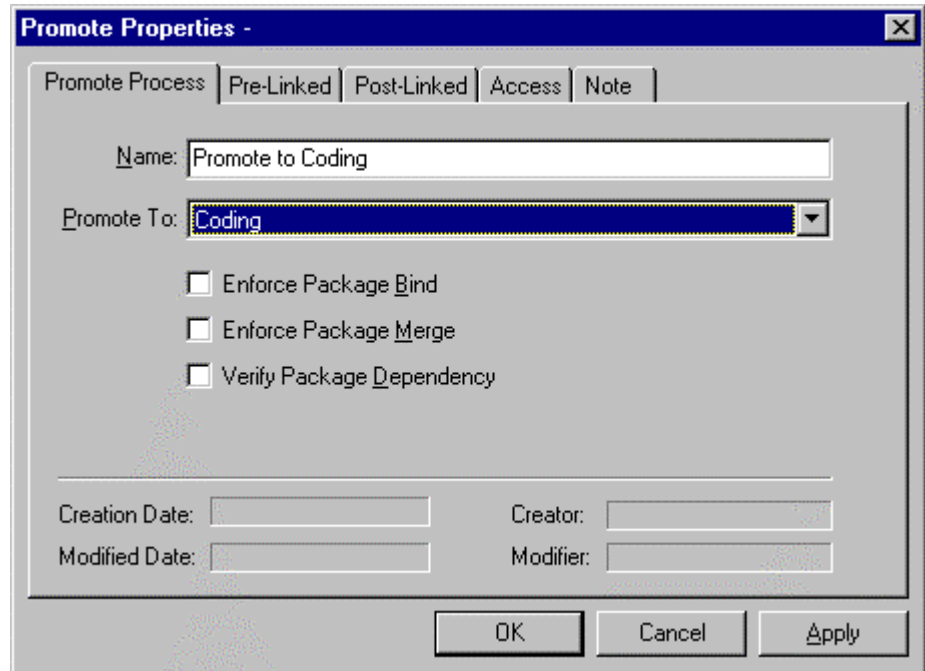
The image shows a Windows-style dialog box titled "Create Package Properties -". It has a tabbed interface with tabs labeled "Create Package Process", "Defaults", "Pre-Linked", "Post-Linked", "Access", and "Note". The "Create Package Process" tab is currently selected. Inside this tab, there is a text field labeled "Name:" containing the text "Create Package". Below it is a dropdown menu labeled "Initial State:" with "Assign" selected. A section titled "Forms" contains a dropdown menu labeled "Type:" with "Problem Report" selected, and a checked checkbox labeled "Create automatically". At the bottom of the tab, there are four text fields: "Creation Date:", "Creator:", "Modified Date:", and "Modifier:". At the very bottom of the dialog are three buttons: "OK", "Cancel", and "Apply".

8. Click the Defaults tab.
9. In the Default Name field, enter **TMR#**. This will cause the name of all packages created to begin with a TMR# prefix.



The image shows the same "Create Package Properties -" dialog box, but now the "Defaults" tab is selected. The "Name:" field is no longer visible. Instead, there is a text field labeled "Default Name:" containing the text "TMR#". The "OK", "Cancel", and "Apply" buttons remain at the bottom.

10. Click OK.
11. Right-click the Processes folder and choose New, Promote Process from the shortcut menu. This will add a Promote process to the Assign state.
12. Enter **Promote to Coding** in the Name field.
13. Choose Coding from the Promote To drop-down list.



The screenshot shows the 'Promote Properties' dialog box with the 'Promote Process' tab active. The 'Name' field is filled with 'Promote to Coding'. The 'Promote To' dropdown menu is open, showing 'Coding' as the selected option. Below these fields are three checkboxes: 'Enforce Package Bind', 'Enforce Package Merge', and 'Verify Package Dependency', all of which are currently unchecked. At the bottom of the dialog, there are four input fields: 'Creation Date', 'Creator', 'Modified Date', and 'Modifier'. To the right of these fields are three buttons: 'OK', 'Cancel', and 'Apply'.

14. Click OK.

## Coding State

1. Click the plus (+) sign next to the Coding state to expand it, and then select the Processes folder.
2. Right-click the Processes folder and choose New, Approve Process from the shortcut menu to add an Approve process to the Coding state.
3. Enter **Approve Package** in the Name field.

4. Click the Add button located at the bottom of the Approval Users list box, select the user JimE and then click OK. Jim should now be listed in the Approval Users list box.

The screenshot shows the 'Approve Properties' dialog box with the following details:

- Title Bar:** Approve Properties -
- Tabs:** Approve (selected), Pre-Linked, Post-Linked, Access, Note
- Name Field:** Approve Package
- Approval Users List:**

Name	Real Name
JimE	Jim Evans
- Approval User Groups List:**

Name
------
- Buttons:** Add, Delete (for both lists); OK, Cancel, Apply (at the bottom)
- Metadata Fields:** Creation Date, Modified Date, Creator, Modifier (all empty)

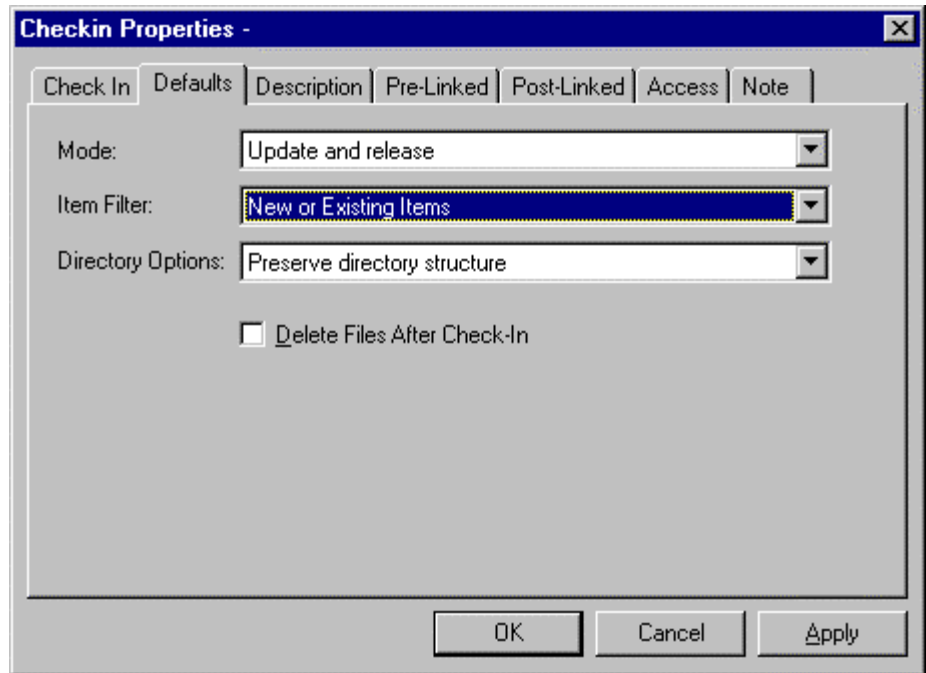
5. Click OK
6. Right-click the Processes folder and choose New, Checkin Process from the shortcut menu to add a Check In process to the Coding state.
7. Enter **Check In Changes** in the Name field.

8. Ensure that only the New or existing items and the Existing items only filters are checked. Deselect New items only.

The screenshot shows the 'Checkin Properties' dialog box with the 'Defaults' tab selected. The 'Name' field contains 'Check In Changes'. The 'Item Filter' section has three checkboxes: 'New or existing items' (checked), 'New items only' (unchecked), and 'Existing items only' (checked). The 'Options' section has one checkbox: 'Check in by owner only' (unchecked). At the bottom, there are fields for 'Creation Date', 'Creator', 'Modified Date', and 'Modifier', all of which are empty. The 'OK', 'Cancel', and 'Apply' buttons are at the bottom right.

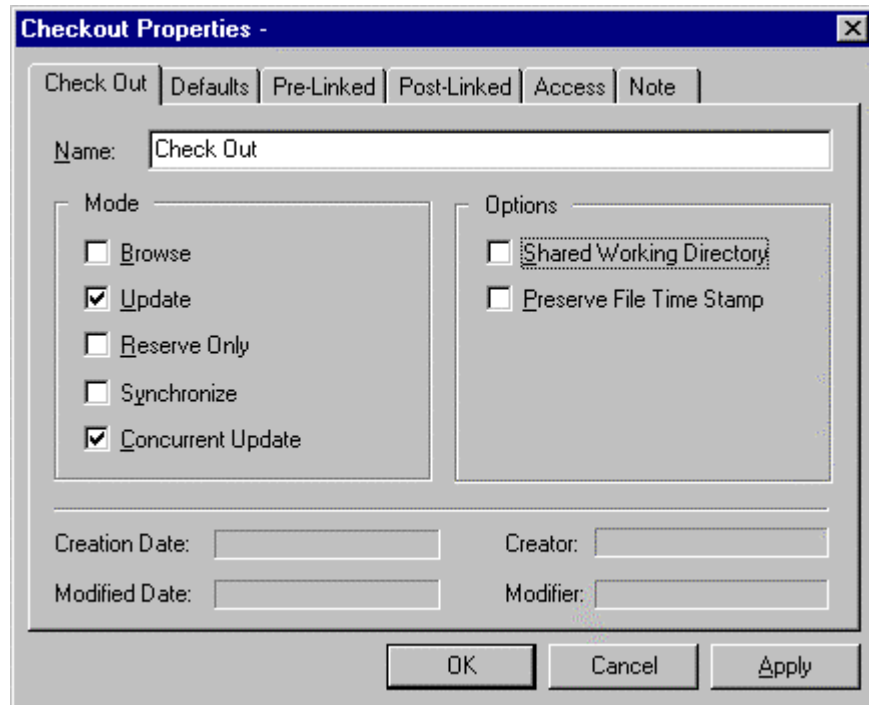
9. Click the Defaults tab.
10. Choose Update and release from the Mode drop-down list.
11. Choose New or Existing Items from the Item Filter drop-down list.
12. Choose Preserve directory structure from the Directory Options drop-down list.

13. Delete Files After Check-In should not be selected; this is the default.



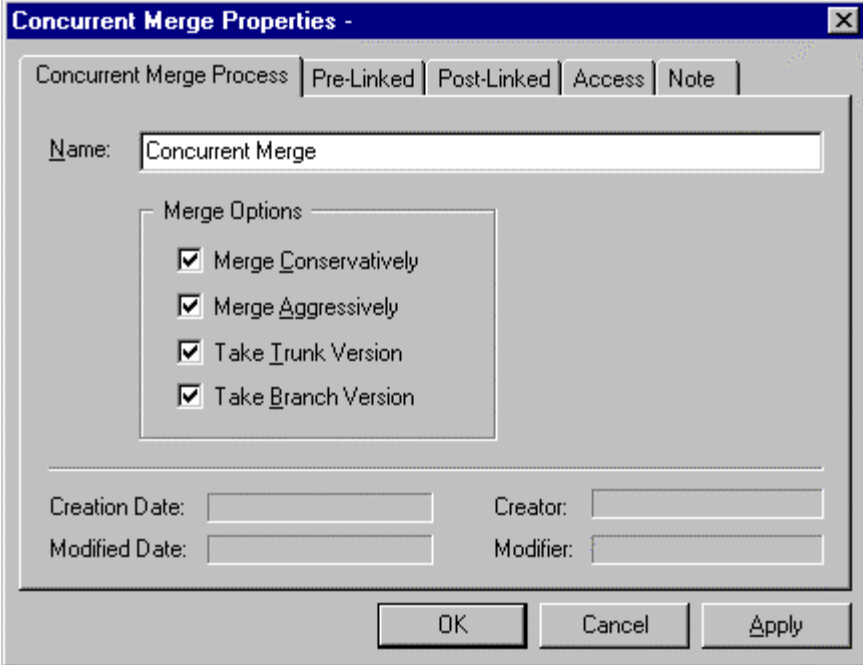
14. Click OK.
15. Right-click and choose New, Checkout Process from the shortcut menu to add a Checkout process to the Coding state.
16. Enter **Check Out** in the Name field.

17. Ensure that the boxes next to the Update and Concurrent Update modes are the only ones checked.



18. Click the Defaults tab and choose Preserve View Path Structure from the Directory Options drop-down list. Deselect the Replace Read Only Files check box.
19. Click OK.
20. Right-click the Processes folder and choose New, Concurrent Merge Process from the shortcut menu to add a Concurrent Merge process to the Coding state.
21. Enter **Concurrent Merge** in the Name field.

22. The Merge Rules check boxes should all be selected.

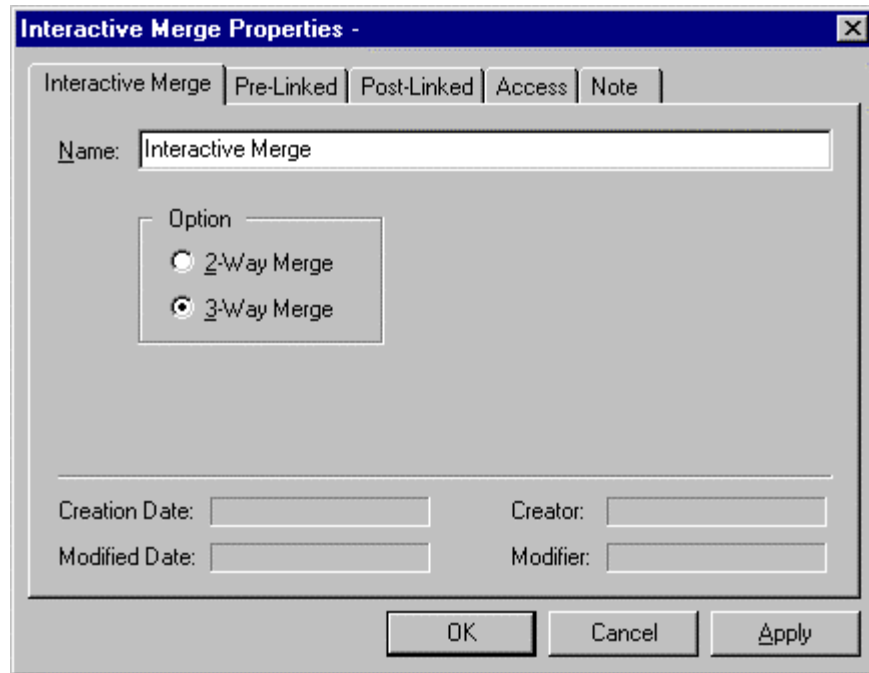


The screenshot shows a dialog box titled "Concurrent Merge Properties -". It has a tabbed interface with four tabs: "Concurrent Merge Process", "Pre-Linked", "Post-Linked", and "Access", with "Concurrent Merge Process" being the active tab. Below the tabs is a "Name:" label followed by a text box containing "Concurrent Merge". Underneath is a section titled "Merge Options" containing four checkboxes, all of which are checked: "Merge Conservatively", "Merge Aggressively", "Take Trunk Version", and "Take Branch Version". At the bottom of the dialog, there are four text boxes for "Creation Date:", "Creator:", "Modified Date:", and "Modifier:". Below these text boxes are three buttons: "OK", "Cancel", and "Apply".

23. Click OK.
24. Right-click the Processes folder and choose New, Interactive Merge Process from the shortcut menu to add an Interactive Merge process to the Coding state.
25. Enter **Interactive Merge** in the Name field.

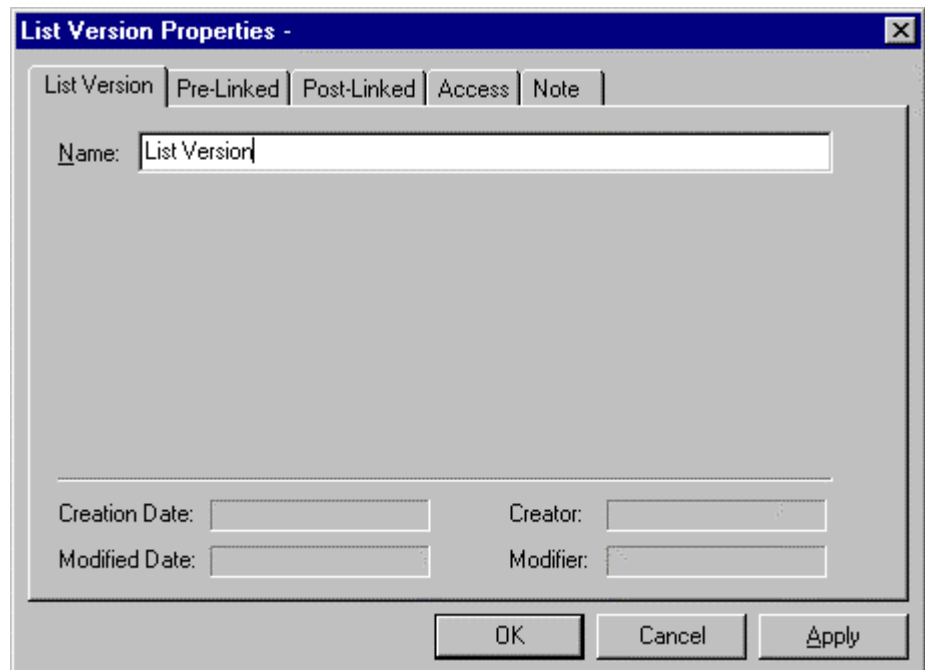


26. The 3-Way Merge option should be selected; this is the default.



The "Interactive Merge Properties" dialog box has a title bar with a close button. It contains five tabs: "Interactive Merge", "Pre-Linked", "Post-Linked", "Access", and "Note". The "Interactive Merge" tab is selected. Inside this tab, there is a "Name:" label followed by a text box containing "Interactive Merge". Below this is an "Option" group box containing two radio buttons: "2-Way Merge" and "3-Way Merge". The "3-Way Merge" radio button is selected. At the bottom of the dialog, there are four text boxes for "Creation Date:", "Creator:", "Modified Date:", and "Modifier:". At the very bottom are three buttons: "OK", "Cancel", and "Apply".

27. Click OK.
28. Right-click the Processes folder and choose New, List Version Process from the shortcut menu to add a List Version process to the Coding state.
29. Enter **List Version** in the Name field.

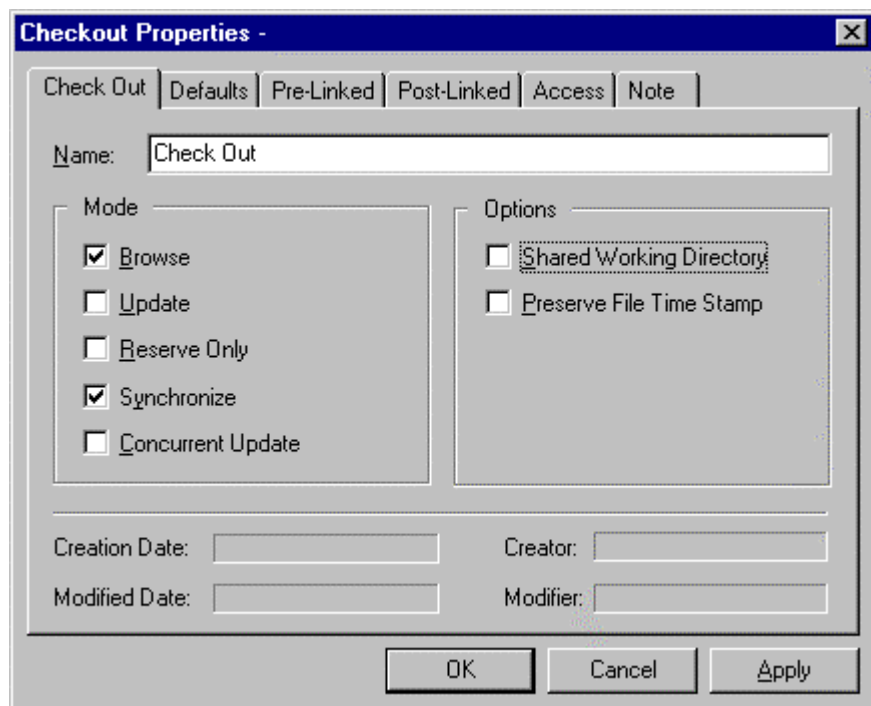


The "List Version Properties" dialog box has a title bar with a close button. It contains five tabs: "List Version", "Pre-Linked", "Post-Linked", "Access", and "Note". The "List Version" tab is selected. Inside this tab, there is a "Name:" label followed by a text box containing "List Version". Below this is a large empty text area. At the bottom of the dialog, there are four text boxes for "Creation Date:", "Creator:", "Modified Date:", and "Modifier:". At the very bottom are three buttons: "OK", "Cancel", and "Apply".

30. Click OK.
31. Right-click the Processes folder and choose New, Promote Process from the shortcut menu to add a Promote process to the Coding state.
32. Enter **Promote to Test** in the Name field.
33. Choose Test from the Promote To drop-down list.
34. Click OK.

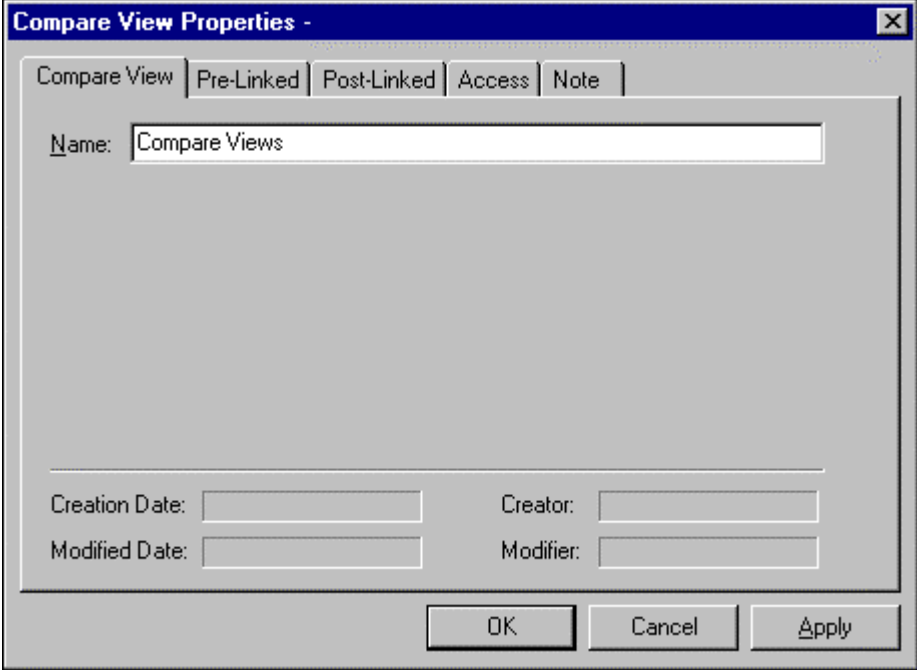
## Test State

1. Click the plus (+) sign next to the Test state to expand it and then right-click the Processes folder.
2. Choose New, Checkout Process from the shortcut menu.
3. Enter **Check Out** in the Name field.
4. Ensure that the check boxes next to the Browse and Synchronize modes are selected. This ensures that versions can be checked out to read, but no changes can be checked back in, and that the versions are up-to-date.
5. Click the Defaults tab and choose Preserve View Path Structure from the Directory Options drop-down list. Deselect the Replace Read Only Files check box.



6. Click OK.

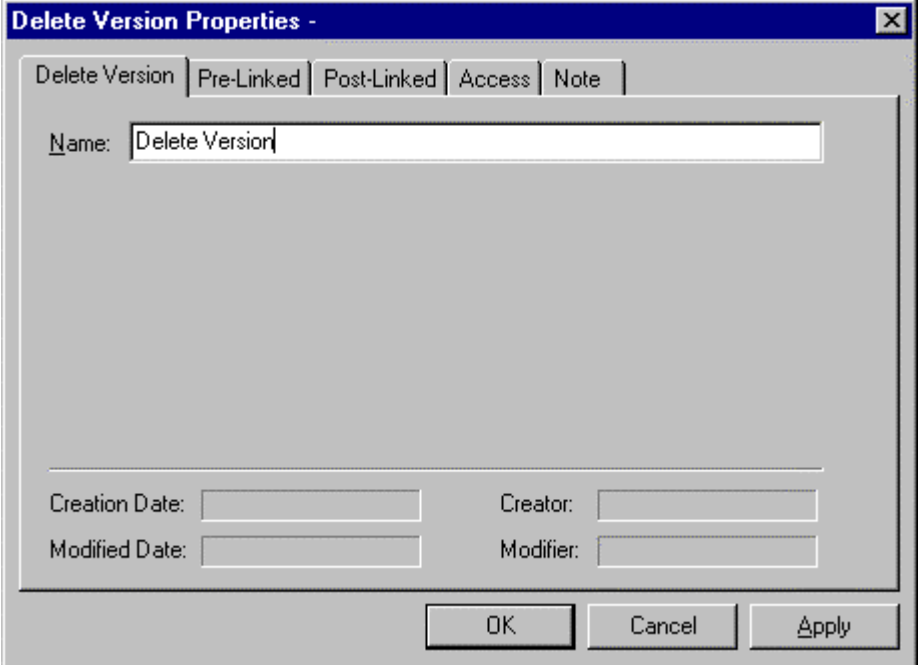
7. Right-click the Processes folder and choose New, Compare View Process from the shortcut menu to add a Compare View process to the Test state.
8. Enter **Compare Views** in the Name field.



The image shows a 'Compare View Properties' dialog box. It has a title bar with a close button. Below the title bar are five tabs: 'Compare View' (selected), 'Pre-Linked', 'Post-Linked', 'Access', and 'Note'. The 'Compare View' tab contains a 'Name:' label followed by a text box containing 'Compare Views'. Below this is a large empty text area. At the bottom of the dialog, there are four input fields: 'Creation Date:', 'Creator:', 'Modified Date:', and 'Modifier:'. At the very bottom are three buttons: 'OK', 'Cancel', and 'Apply'.

9. Click OK.
10. Right-click the Processes folder and choose New, Delete Version Process from the shortcut menu to add a Delete Version process to the Test state.

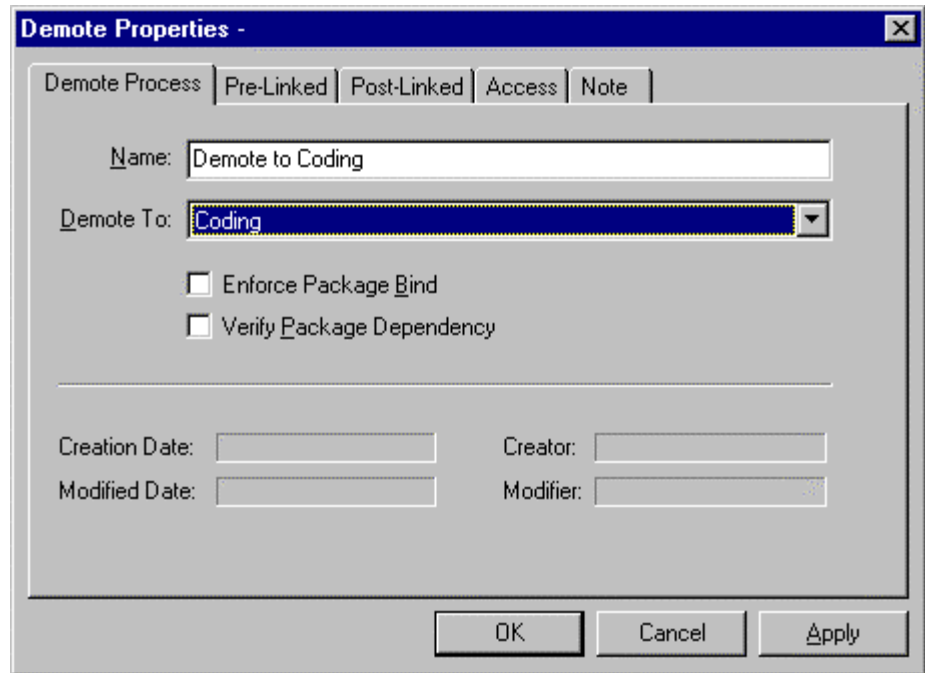
11. Enter **Delete Version** in the Name field.



The image shows a Windows-style dialog box titled "Delete Version Properties -". It has a tabbed interface with five tabs: "Delete Version", "Pre-Linked", "Post-Linked", "Access", and "Note". The "Delete Version" tab is currently selected. Inside the dialog, there is a text field labeled "Name:" containing the text "Delete Version". Below this, there are four more text fields arranged in two rows: "Creation Date:", "Creator:", "Modified Date:", and "Modifier:". At the bottom right of the dialog, there are three buttons: "OK", "Cancel", and "Apply".

12. Click OK.
13. Right-click the Processes folder and choose New, Demote Process from the shortcut menu to add a Demote process to the Test state.
14. Enter **Demote to Coding** in the Name field.

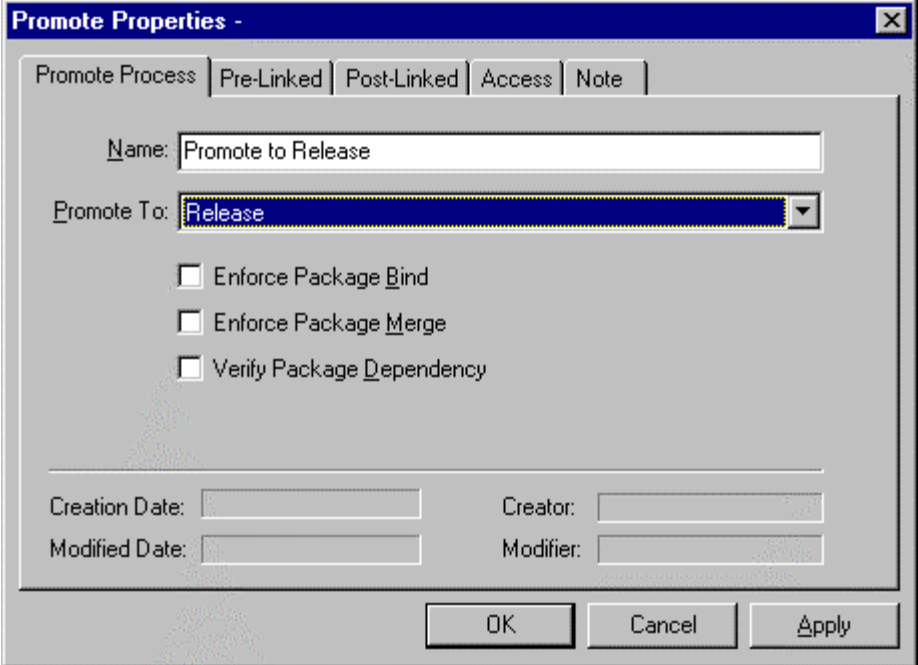
15. Choose Coding from the Demote To drop-down list.



The screenshot shows a dialog box titled "Demote Properties -". It has a tabbed interface with tabs for "Demote Process", "Pre-Linked", "Post-Linked", "Access", and "Note". The "Demote Process" tab is selected. Inside the dialog, there is a "Name:" text box containing "Demote to Coding". Below it is a "Demote To:" drop-down menu with "Coding" selected. There are two checkboxes: "Enforce Package Bind" and "Verify Package Dependency", both of which are unchecked. At the bottom, there are four text boxes for "Creation Date:", "Creator:", "Modified Date:", and "Modifier:". At the very bottom of the dialog are three buttons: "OK", "Cancel", and "Apply".

16. Click OK.
17. Right-click the Processes folder and choose New, Promote Process from the shortcut menu to add a Promote process to the Test state.
18. Enter **Promote to Release** in the Name field.

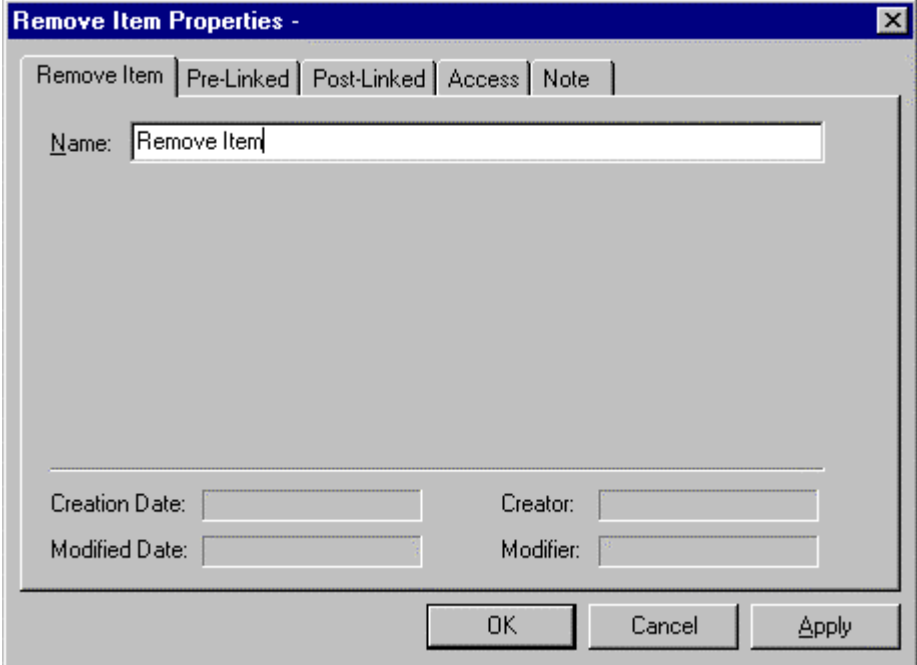
19. Choose Release from the Promote To drop-down list.



The image shows a 'Promote Properties' dialog box with a blue title bar and a close button. It has five tabs: 'Promote Process', 'Pre-Linked', 'Post-Linked', 'Access', and 'Note'. The 'Promote Process' tab is selected. Inside the dialog, there is a 'Name:' text box containing 'Promote to Release'. Below it is a 'Promote To:' drop-down menu with 'Release' selected. There are three unchecked checkboxes: 'Enforce Package Bind', 'Enforce Package Merge', and 'Verify Package Dependency'. At the bottom, there are four text boxes for 'Creation Date', 'Creator', 'Modified Date', and 'Modifier'. At the very bottom are three buttons: 'OK', 'Cancel', and 'Apply'.

20. Click OK.
21. Right-click the Processes folder and choose New, Remove Item Process from the shortcut menu to add a Remove Item process to the Test state.

22. Enter **Remove Item** in the Name field.



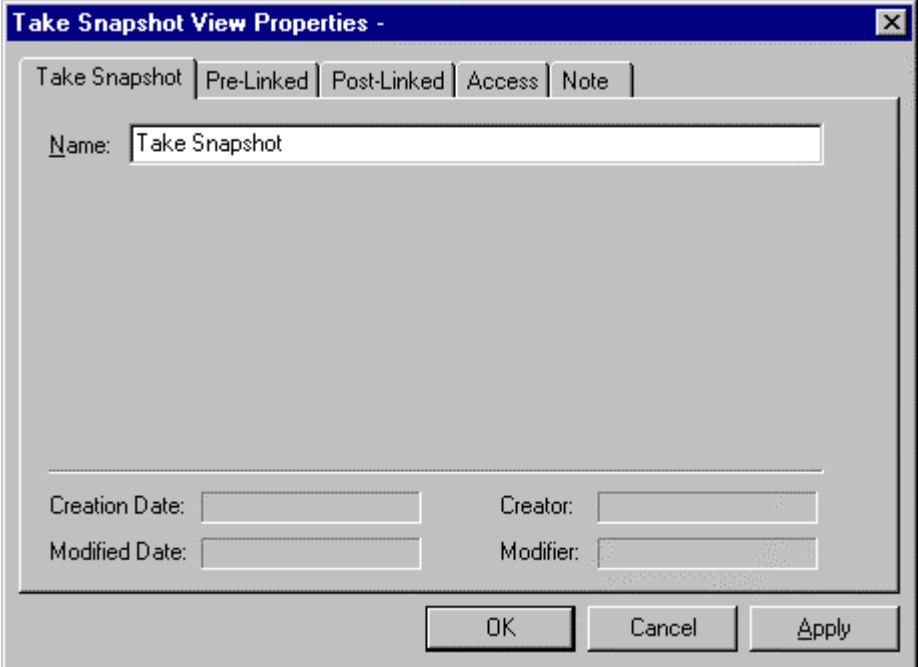
The screenshot shows a dialog box titled "Remove Item Properties -". It has five tabs: "Remove Item", "Pre-Linked", "Post-Linked", "Access", and "Note". The "Remove Item" tab is active. Inside the dialog, there is a "Name:" label followed by a text box containing "Remove Item". Below this, there are four more fields: "Creation Date:", "Creator:", "Modified Date:", and "Modifier:", each followed by an empty text box. At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Apply".

23. Click OK.

## Release State

1. Click the plus (+) sign next to the Release state to expand it, and then select the Processes folder.
2. Right-click and choose New, Take Snapshot View Process from the shortcut menu to add a Take Snapshot View process to the Release state.

3. Enter **Take Snapshot** in the Name field.



The image shows a dialog box titled "Take Snapshot View Properties -". It has a tabbed interface with five tabs: "Take Snapshot", "Pre-Linked", "Post-Linked", "Access", and "Note". The "Take Snapshot" tab is currently selected. Inside the dialog, there is a large text area labeled "Name:" containing the text "Take Snapshot". Below this, there are four input fields arranged in two rows: "Creation Date:", "Creator:", "Modified Date:", and "Modifier:". At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Apply".

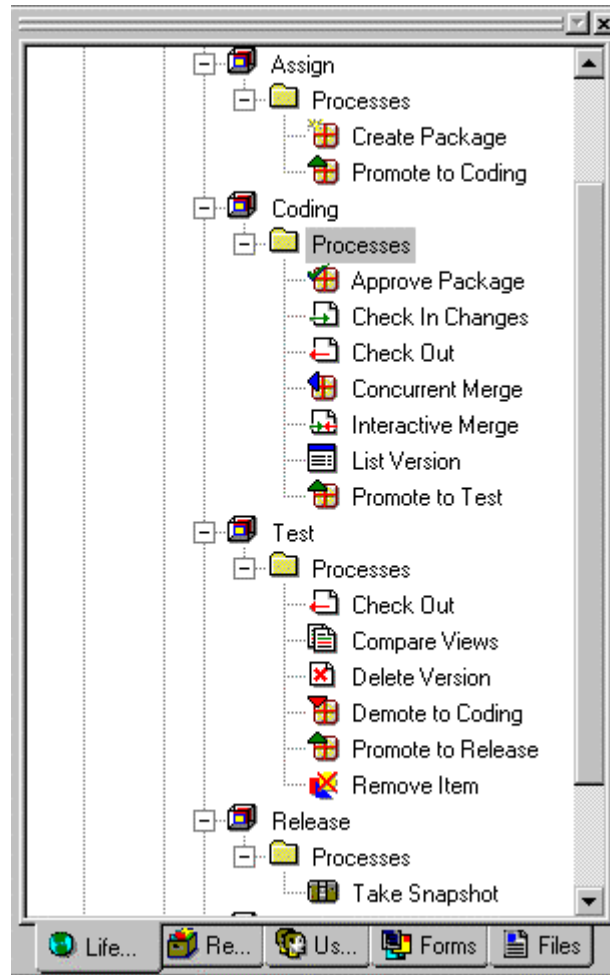
4. Click OK.



## Snapshot State

You do not need to add processes to the Snapshot state. You will use this state later as a holding place for your snapshots.

When you are finished, the life cycle tree in the workspace for the Tutorial 1.0 Project should look like the following:

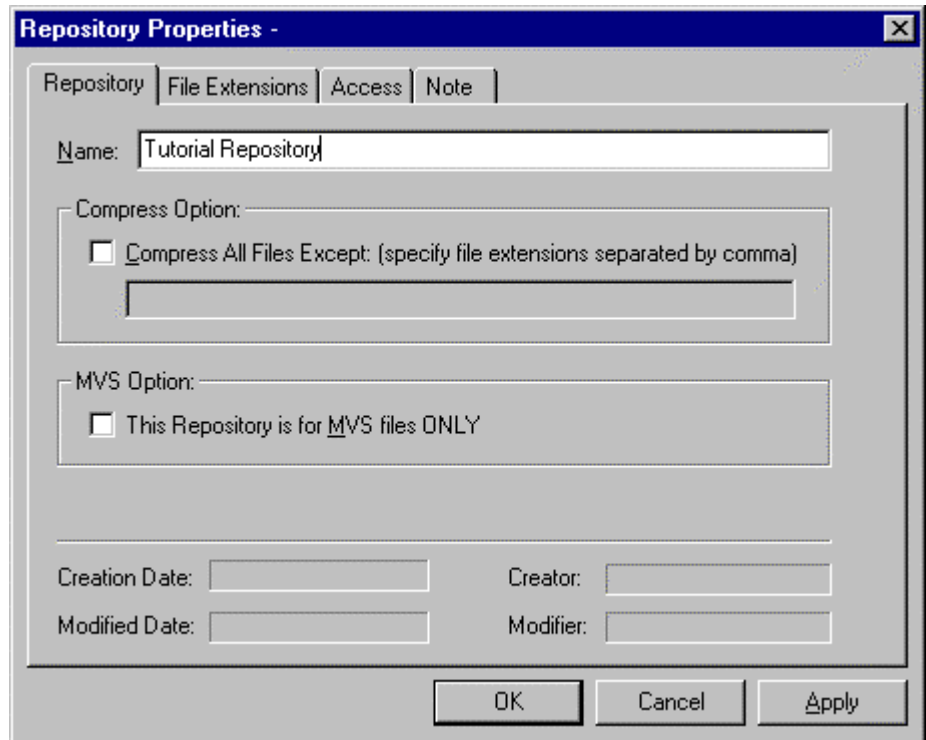


## Step 7: Create a Repository

A repository is where all of the files that are under Harvest's control are stored. Repository data is stored inside tables in the relational database.

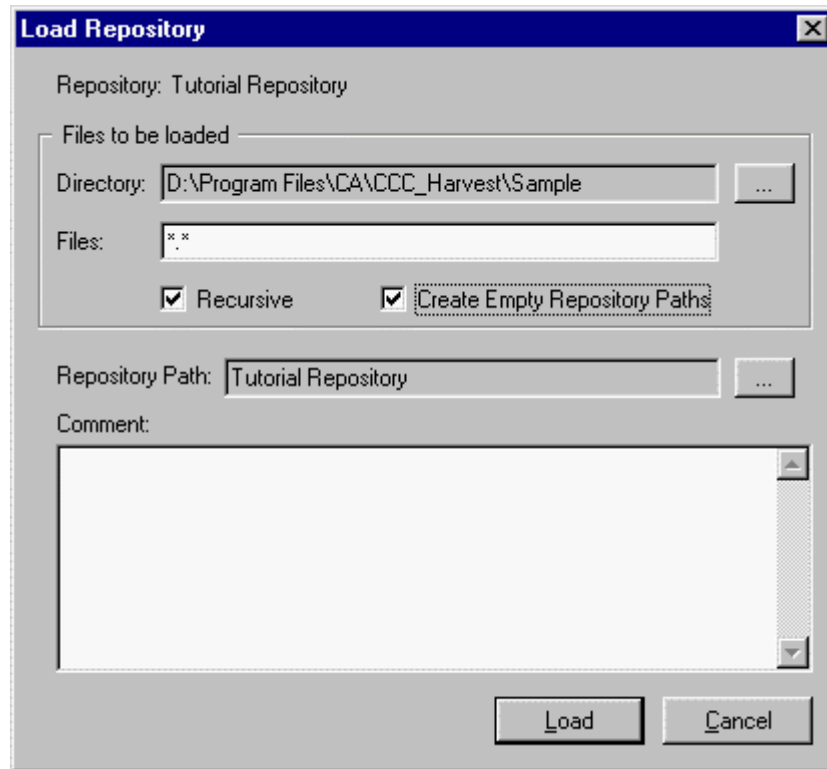
1. Select the Repositories tab at the bottom of the workspace to display a list of existing repositories. If you are using a new installation of Harvest, you will probably not see any repositories listed.

2. Right-click the server you want and choose New Repository from the shortcut menu. The Repository Properties dialog will be displayed.
3. Enter **Tutorial Repository** in the Name field.
4. The Compress Option and MVS Option should not be selected; this is the default.



5. Click OK.
6. Right-click Tutorial Repository and choose Load Repository from the shortcut menu. The Load Repository dialog will appear.
7. Click the button next to the Directory field.
8. Navigate to your %HARVESTHOME%\Sample directory, select it and then click OK.
9. Click both the Recursive and the Create Empty Repository Paths check boxes to enable them.
10. Tutorial Repository should be displayed in the Repository Path field. If it is not, click the button next to the Repository Path field.

11. From the Select a Repository Path, select Tutorial Repository and then click OK. Your Load Repository dialog should now look similar to the following:



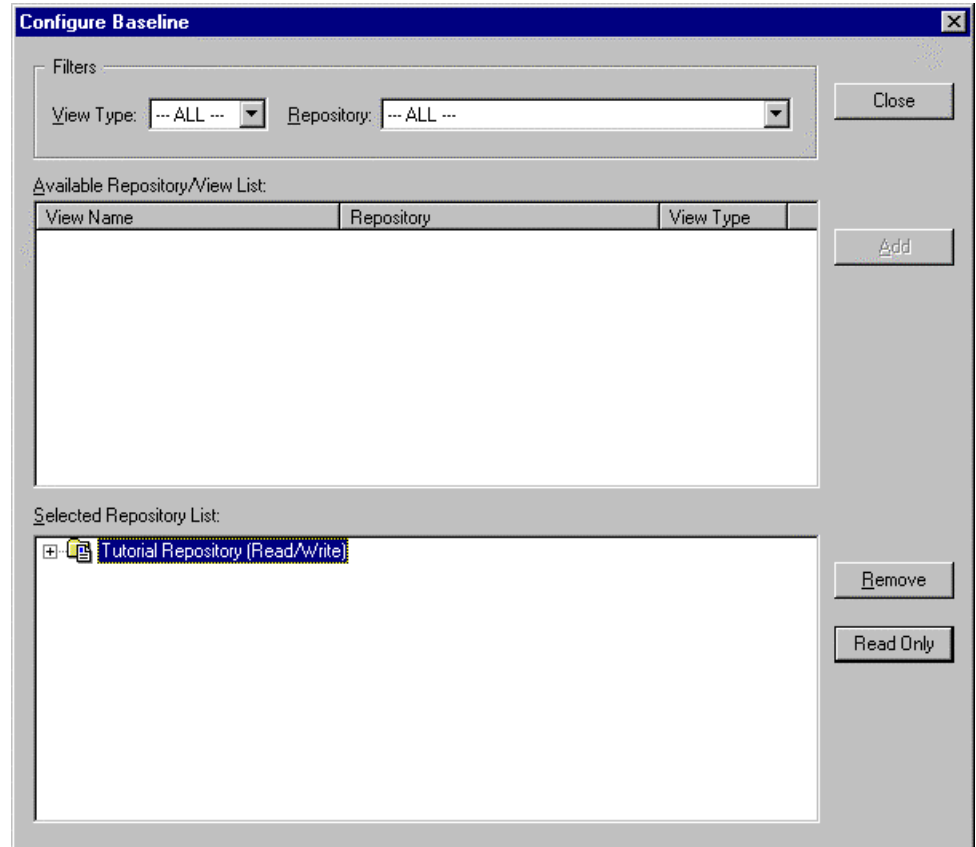
11. Click Load to execute the load and to close the dialog. A meter located at the bottom of the main window shows the progress of the load process.
12. Check the output log to verify the files were successfully loaded.

## Step 8: Establish the Baseline

A baseline defines the items that are available to users of the project. A baseline can include items from one or more repositories. In this exercise, you will only include one repository in the Baseline.

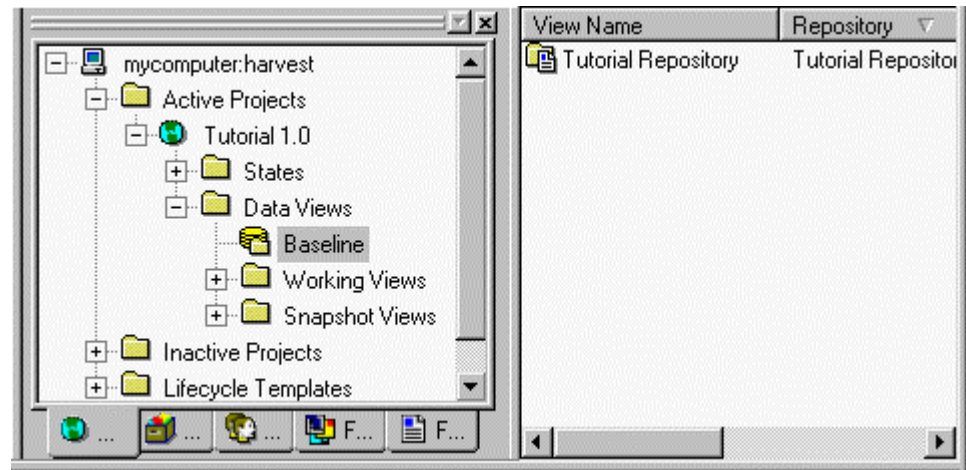
1. Click the Lifecycles tab.
2. Click the plus (+) sign next to the Data Views to expand it, right-click Baseline in the Data Views folder of the Tutorial 1.0 project and choose Configure Baseline from the shortcut menu. The Configure Baseline dialog will appear.
3. You will see Tutorial Repository listed in the Available Repository/View List. Select Tutorial Repository and click Add. Click Yes on the confirmation dialog.

4. Tutorial Repository is now listed in the Selected Repository List as Read Only. Select Tutorial Repository and click the Read/Write button. Click Yes on the confirmation dialog.



5. Click Close.

The Administrator window workspace and list view should now look like the following:



## Step 9: Set the Access Permissions

For every Harvest object there are varying access permissions that can be set. Access information is comprised of two parts:

- method (action that can be taken)
- user group (who can perform the action)

Using an object's properties dialog, access is granted to one or more user groups, allowing members of those groups to perform a particular kind of action (method) on the object. For example, at the Harvest-level access could be granted to the group Repository Manager allowing members to administer repositories, or at the process level, the group Developer could be given access to check in and check out files.

There are eight objects within Harvest that can be secured through access control: Harvest, projects, states, processes, repositories, item paths, items, and form types. Other Harvest objects are secured through an object to which they belong. For example, views, packages, and package groups are all secured through the project or state of which they are a part.

To ensure that the user groups you created have Use Access to the Tutorial 1.0 project, you need to verify that Use Access has been set for the Public user group. To verify this, perform the following steps:

1. Right-click the Tutorial 1.0 project folder and choose Properties from the shortcut menu.
2. Select the Access tab and from the Access Type drop-down list, choose Use Access.
3. Verify that the user group Public is listed as User Groups with Access.
4. If not, click the Add button to get the list of available User Groups.
5. Select Public and click OK to add this user group.
6. Click OK to save and exit the Project Properties dialog.

To give Update Access to the Tutorial 1.0 project you have created, follow these steps:

1. Under the Lifecycle tab, expand and navigate to the Tutorial 1.0 project folder.
2. Right-click the Tutorial 1.0 project folder and select Properties.
3. Select the Access tab and from the Access Type drop-down list choose Update Access.
4. Click the Add button to list the available user groups.
5. Select the Tutorial Dev. Manager user group and click OK. Tutorial Dev. Manager will be listed in the User Groups with Access box. Click OK to give the Tutorial Dev. Manager user group update access.
6. Click OK to save and exit the Project Properties dialog.

To give Update Package Access to packages in the states, follow these steps:

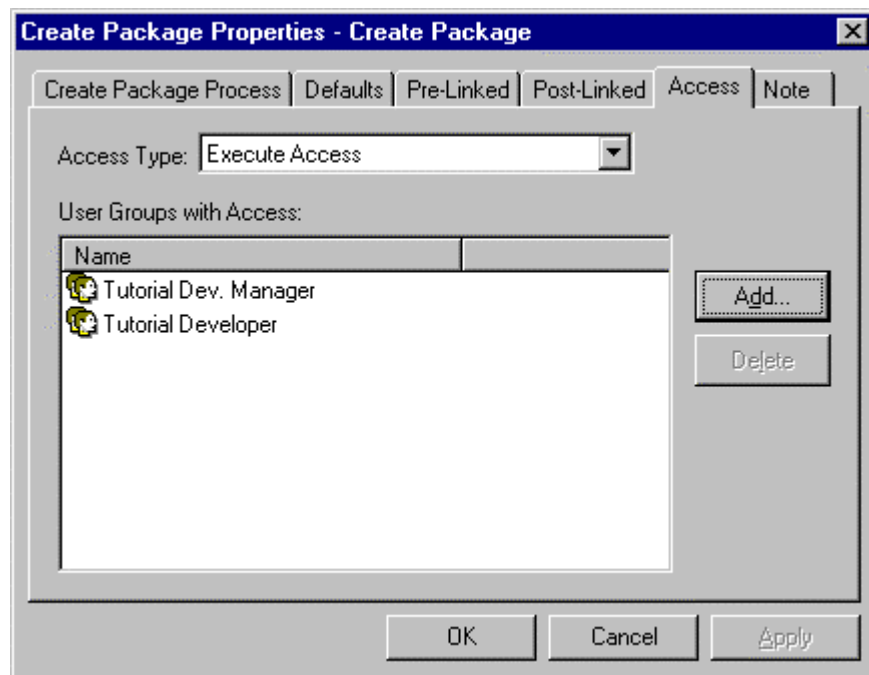
1. Under the Lifecycle tab, expand and navigate to the Tutorial project folder.
2. Expand the Tutorial project folder and navigate to the Coding state folder.
3. Right-click the Coding state folder and choose Properties from the shortcut menu.
4. Select the Access tab and from the Access Type drop-down list choose Update Package Access.
5. Click the Add button to list the available user groups.
6. Select the Tutorial Dev user group and click OK. This lists Tutorial Dev in the User Groups with Access box. Click OK to give the Tutorial Dev user group Update Package Access.

7. Navigate to the Test state and using steps 1 through 6 give Update Package Access to the Tutorial QA user group.

**Note:** Without Update Package Access, a user does not have access to update forms. Without this type of access the user cannot edit forms and would be unable to pass vital information to the next stage of the development life cycle.

To give execute access to every process you created in Step 3, follow these steps.

1. In the Tutorial 1.0 project, navigate to the Assign state and expand it.
2. Double-click on the Processes folder.
3. In the list view, right-click the Create Package process and choose Properties from the shortcut menu.
4. On the Create Package Properties dialog, click the Access tab.



5. Choose Execute Access from the Access Type drop-down list.
6. Click the Add button and choose Tutorial Developer and Tutorial Dev. Manager from the User Groups list. Click OK.
7. Click OK to save and exit the Create Package Properties dialog.

To assign access to the remaining processes, follow the same steps you used for the create package process. The table below shows the information you will need to complete your access setup.

When you finish the Assign state:

1. Collapse the Assign state tree and navigate to the Coding state.
2. Follow the same steps as above to set the access for the Coding, Test and the Release states.

State	Process Name	User Group
Assign	Create Package	Tutorial Developer
		Tutorial Dev. Manager
Assign	Promote to Coding	Tutorial Dev. Manager
Coding	Approve Package	Tutorial Dev. Manager
Coding	Check In Changes	Tutorial Developer
Coding	Check Out	Tutorial Developer
Coding	Concurrent Merge	Tutorial Developer
Coding	Interactive Merge	Tutorial Developer
Coding	List Version	Tutorial Dev. Manager
Coding	Promote to Test	Tutorial Dev. Manager
Test	Check Out	Tutorial QA
Test	Compare Views	Tutorial QA
Test	Delete Version	Tutorial QA
Test	Demote to Coding	Tutorial QA
Test	Promote to Release	Tutorial QA
Test	Remove Item	Tutorial QA
Release	Take Snapshot	Tutorial Dev. Manager

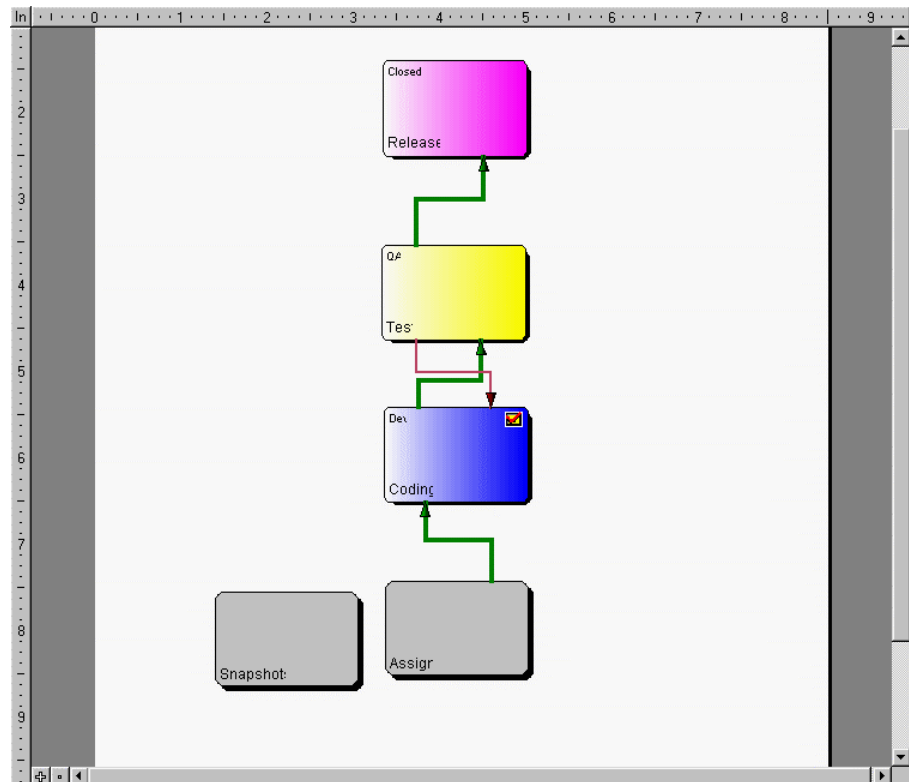
In this example, only the Tutorial Developer has more than one user name. In your organization, you will likely find there are many users in some user groups and only a few in others. Assigning access by user group obviously saves time and makes maintenance much easier.



## Step 10: Review the Life Cycle

Now that you have set up your own Harvest project, you can view a graphical representation of it by invoking the Lifecycle Viewer.

1. Right-click the Tutorial 1.0 project and choose View Project Lifecycle from the shortcut menu.
2. The Lifecycle Viewer will appear in the list view. Expand the Lifecycle Diagram to fill the List view area. Move the scroll bars around until you see a box labeled Assign.



3. You will also notice lines connecting some of the boxes. Each line represents promote and demote processes between the states. Because packages cannot be demoted from the Coding or the Release states, they are not bi-directional lines. You can invoke the promote and demote properties dialogs by double-clicking on the respective lines.
4. The Lifecycle Viewer allows you to rearrange the way the boxes appear on your screen. Drag the Test box down a few inches. Notice how the lines connecting the boxes move as you move the box.
5. Right-click in any location the diagram to open a shortcut menu with options such as Optimize Layout and Zoom.

6. You can invoke the State Properties dialog from the Lifecycle Viewer by double-clicking a state or by right-clicking a state and choosing Properties from the shortcut menu. When you invoke the State Properties dialog from the Lifecycle Viewer, you still have access to all the capabilities you had when you accessed the State Properties from the Administrator window. If you needed to make changes in your state, you could make them here.
7. Double-click on the Test box and to open the Properties dialog. You will not be making changes at this point.
8. Select Cancel on the State Properties dialog.

## Step 11: Creating a Life Cycle Template

In order to use the Tutorial 1.0 project as a template for later projects, you need to copy the Tutorial 1.0 project to the Lifecycle Templates folder.

1. Right-click the Tutorial 1.0 project folder and choose Copy To from the shortcut menu.
2. In the Copy Project dialog, type in Tutorial Template in the Project Name field.
3. If Lifecycle Templates is not showing in the Copy To field, choose Lifecycle Templates from the Copy To drop-down list.
4. Accept the default, Duplicate Access Control. Click OK.
5. After the Copy To process is finished, expand the Lifecycle Templates folder and verify that the Tutorial Template is listed. The project you have created is now ready to use as a template for future projects and will be used later in this tutorial.
6. Exit Harvest by choosing File, Exit from the main window menu.

## Let's Review

The process you just followed created a completely functional Harvest installation. To review the steps:

1. You began by creating three user groups.
2. You created four users and assigned them to their respective user groups.
3. You created a project named Tutorial 1.0.
4. You defined working views so that only the correct versions of the software items will be visible and accessible to the individuals working in each state.
5. You added states to the project to match the phases of your software development process.

6. You defined the processes that can be executed from each state and created a package template to help maintain a standard naming convention for the change request packages.
7. You created a repository on the Harvest server and then loaded the application files into it.
8. You set the baseline for the project.
9. Finally, you went through every Harvest process, in each state, giving access to the user groups that will use each process.
10. You reviewed your life cycle using the Lifecycle Viewer.



# Using a Harvest Project

---

In this chapter you will explore how the Tutorial 1.0 project you set up in Chapter 3 appears to the end user. You will also explore the world of the development team as they use the Tutorial 1.0 project on a day-to-day basis.

It is **very important** that you complete the steps in this chapter in the order they are specified. Skipping steps can result in errors.

## Instructions for This Chapter

You should follow the exercises in this chapter while using your client machine. You must have completed Chapter 3 successfully before beginning this chapter.

You will log in and out of the Harvest workbench using the Harvest user names you created in Chapter 3.

## Background Information

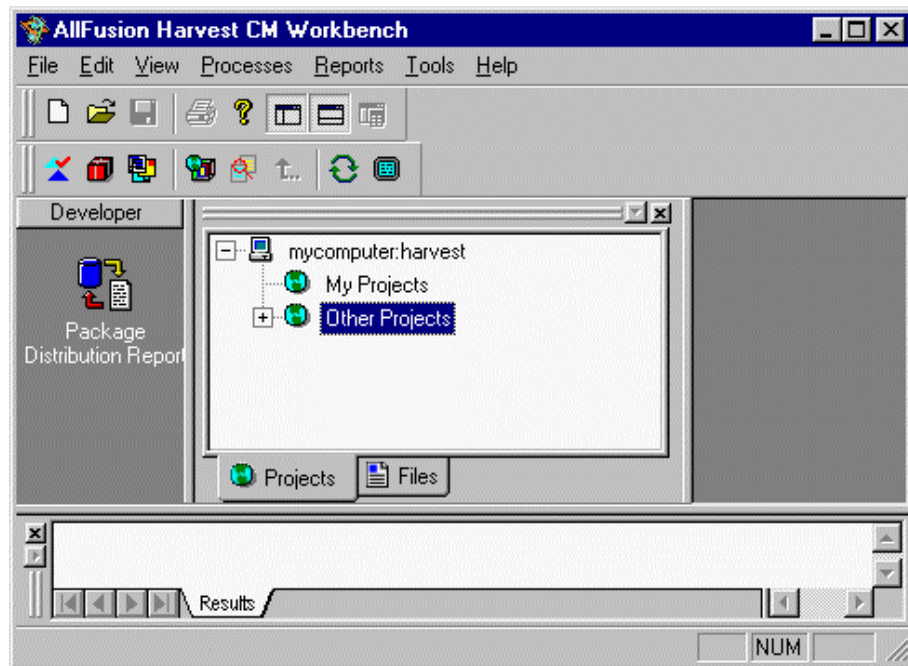
The exercises in this chapter will take you through a typical development scenario. You will perform the various functions posing as different users because each user has different responsibilities and privileges. The scenario you will be following is summarized below.

1. Jim, the development manager, finds a bug. He creates a change request package, assigns it to Bob and promotes it to the Coding state so Bob can work on it.
2. Bob checks out the affected source code item, fixes the bug, and then checks the file back in.
3. Jim approves the changes.
4. Rick promotes the package to the Test state so he can test the fix.
5. The modification does not pass Rick's tests, so he demotes the package back to the Coding state for Bob to fix.

## Step 1: Create a Package

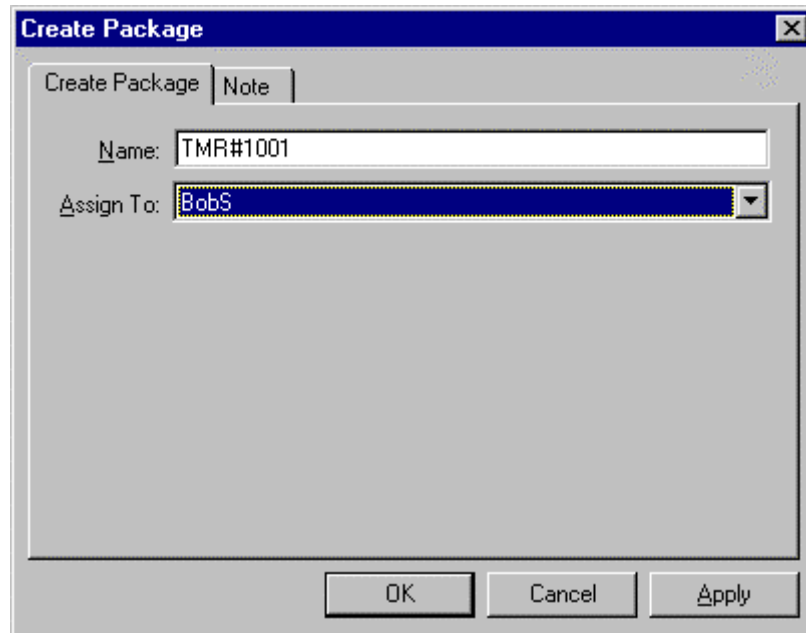
Jim has found a bug in the software and he wants to open a change request to be tracked in Harvest. He will need to create a new package in the Assign state and enter a description of the bug in a form associated with the package.

1. Log in to the workbench to the workbench as JimE.
2. Click Cancel on the Default Context dialog. You will see the following window:

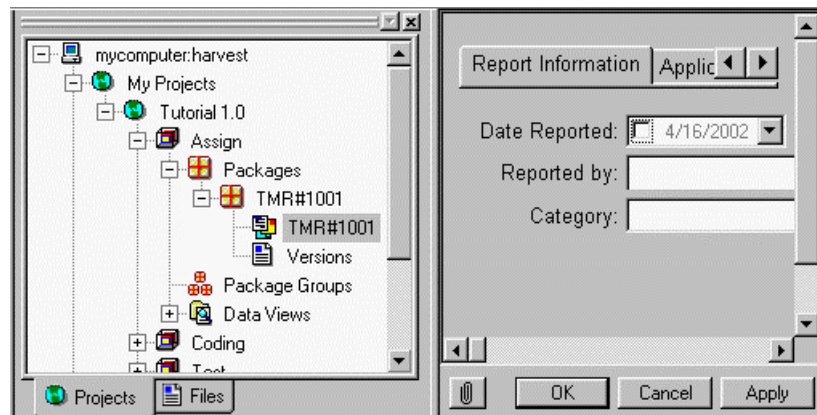


3. On the Projects tab, expand the Other Projects folder. Right-click Tutorial 1.0 and choose Move to "My Projects" folder from the shortcut menu. By moving the Tutorial 1.0 project to your My Projects folder, you can filter out those projects you are not currently working on.
4. Expand the folder My Projects by clicking on the plus (+) sign to the left of it.
5. Expand the Tutorial 1.0 project.
6. Select the Assign state.
7. Right-click the Assign state and choose Processes, Create Package from the shortcut menu or choose Processes, Create Package from the main window menu. This will open the Create Package process execution dialog.
8. Change the contents of the Name field to **TMR#1001**.

9. In the Assigned To field, choose **BobS** from the drop-down list.



10. Click OK to create the package.
11. On the workbench, expand the Assign state and the Packages folder, and then click on the package TMR#1001. You should see a form named TMR#1001 listed. This form was automatically created and associated with the new package because that is how you set up the create package process in the previous chapter.



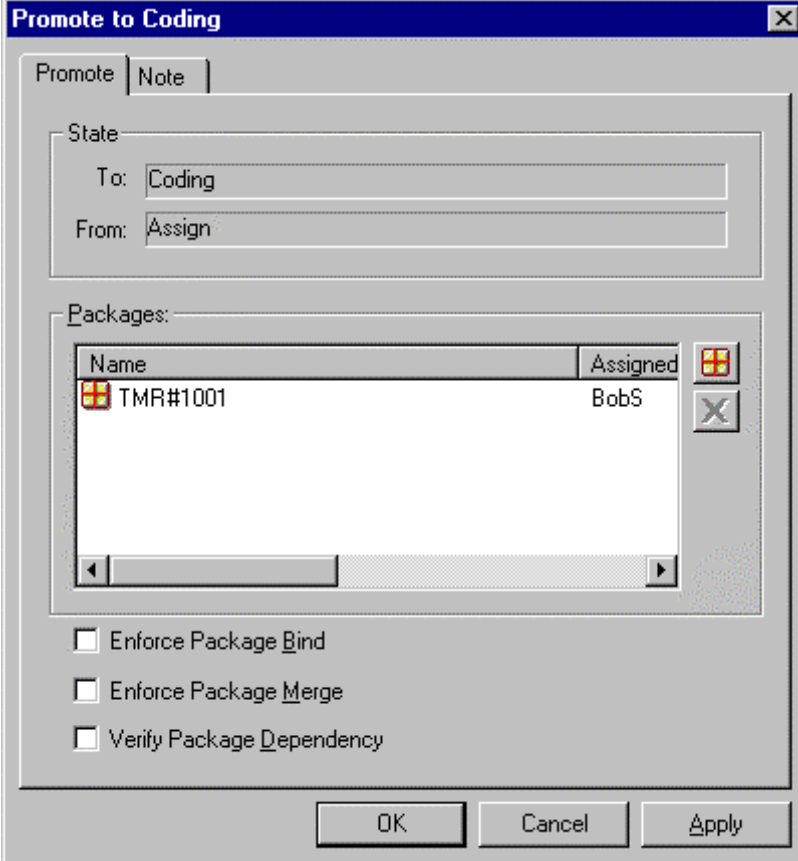
12. In the list view, notice that the Date Reported field is already filled in. This is because this form type has a SQL database trigger associated with it that automatically fills this information in. You can enter field information.
13. Click the box in the field next to the date to accept the shown date. If you do not click this box, the date will change depending on the actual calendar date.

14. Click the OK button on the form to save your changes and close the form.

## Step 2: Promote the Package to Coding

Because Jim is the Development Manager, he has the authority to promote the new package to the Coding state so it can be fixed.

1. Right-click the package TMR#1001 and choose Processes, Promote to Coding from the shortcut menu. The following dialog will appear:



The dialog box titled "Promote to Coding" has two tabs: "Promote" (selected) and "Note". Under the "Promote" tab, there are two text fields: "To:" with the value "Coding" and "From:" with the value "Assign". Below these is a section labeled "Packages:" containing a table with two columns: "Name" and "Assigned". The table lists one package, "TMR#1001", assigned to "BobS". To the right of the table are two icons: a plus sign in a square and a minus sign in a square. Below the table are three checkboxes: "Enforce Package Bind", "Enforce Package Merge", and "Verify Package Dependency", all of which are currently unchecked. At the bottom of the dialog are three buttons: "OK", "Cancel", and "Apply".

Name	Assigned
TMR#1001	BobS

2. Click OK to confirm that you want to promote the package to the Coding state.

Notice that the package no longer exists in the Assign state.

3. Click the plus (+) sign next to the Coding state. The package TMR#1001 should be listed there now.

Jim is done for now, so go ahead and log out of Harvest by choosing File, Exit from the main window menu.

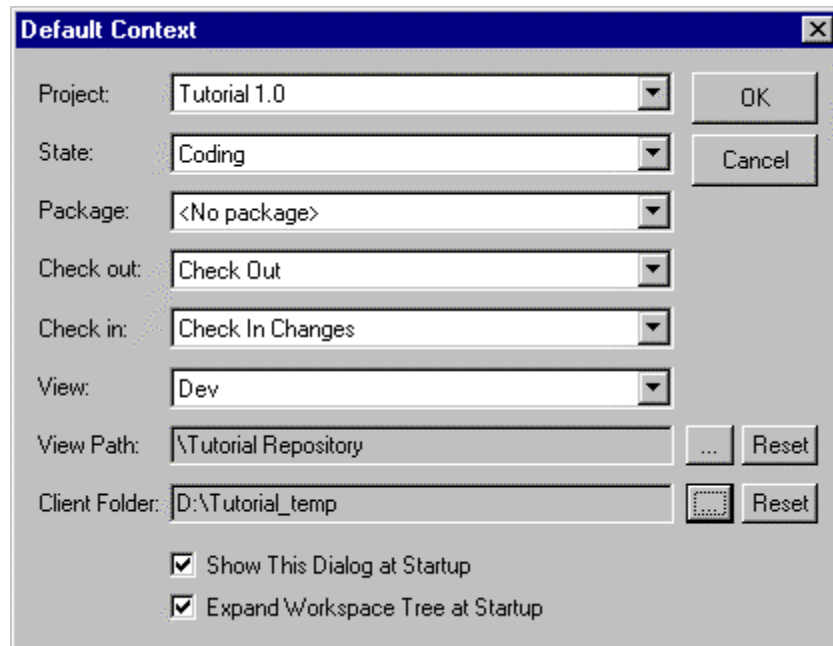


## Step 3: Check Out an Item For Update

In Step 1, Jim assigned Bob to fix the bug. Bob is ready to check out the affected file so that he can make the necessary changes.

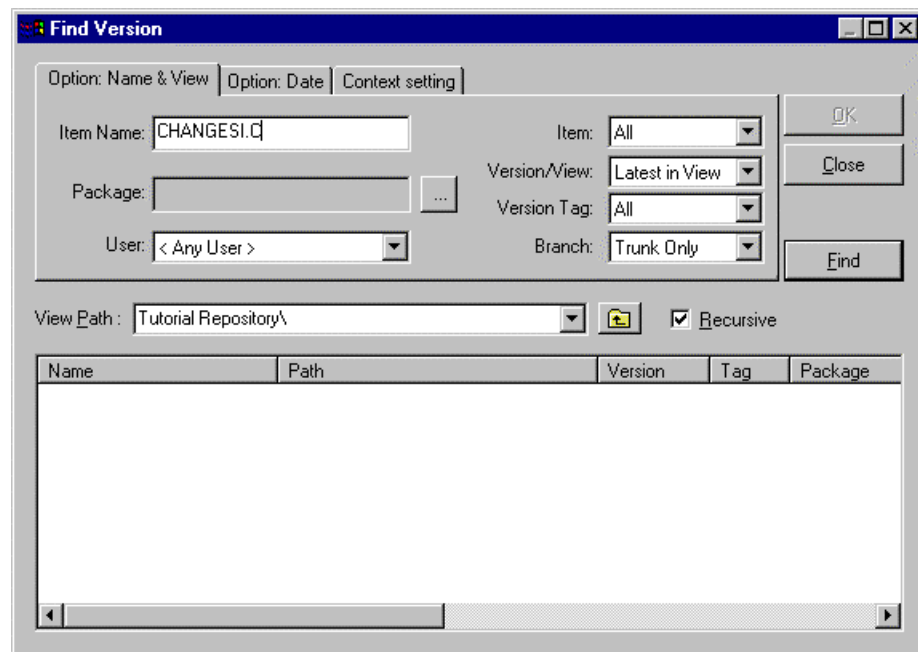
1. Log in to the workbench as BobS.
2. On the Default Context dialog, use the drop-down lists to choose the Tutorial 1.0 project, Coding state, check out and check in processes, and Dev view.

In order for Harvest to synchronize the location of the file after you have checked it out to your filesystem and the location of the item within the repository, you must specify the View Path and Client Folder locations. Click the buttons next to the path fields to browse for and set the view path and your client folder (the working directory you set up in Chapter 2, Step 4: Create a Working Directory). Click OK.



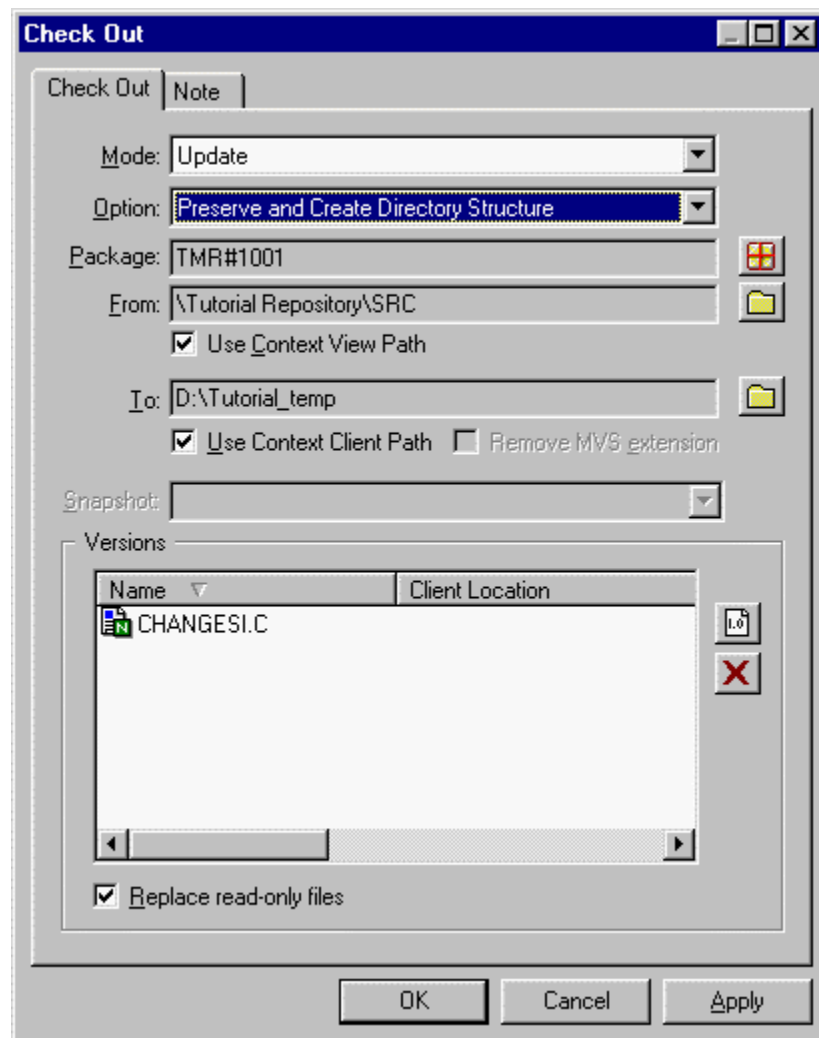
3. Click on the Find Version icon (two triangles with a check mark) on the workbench toolbar.
4. Enter CHANGES.I.C in the Item Name field.

At this point, there are no versions associated with package TMR#1001. BobS can see package TMR#1001 in the Package field drop-down list. But if this empty package is added to the context setting, it will be used as a filtering value and Find Versions (steps 3-5) will show no results. Do **not** select a package as part of the Default Context.



5. Check the Recursive box to enable a recursive search and click the Find button. CHANGESI.C will be listed in the search results field.
6. Select and right-click the item CHANGESI.C and choose Check Out from the shortcut menu. Notice that CHANGESI.C is displayed in the Versions list box of the check out dialog automatically.
7. The Mode drop-down list should be set to Update because you will be making changes to an item.
8. Change the Option drop-down list to Preserve and Create Directory Structure. This will create a mirror image of the directory structure where CHANGESI.C resides within the repository on your local filesystem.
9. All changes to items must be associated with a change package because the package moves through the life cycle, bringing the associated changes with it. Ensure that the package TMR#1001 is shown in the Package field, if not click the button next to the field to locate and select TMR1001.
10. The To and From fields should show the locations you specified in the Default Context dialog. If not, select your default contexts by selecting the Use Context checkboxes located beneath each field.

Your Check Out dialog should now look similar to the following:



15. Click the OK button to execute the check out.

## Updating the File

Now that the file is checked out, Bob can make the changes. Fortunately, the change he needs to make is a simple one.

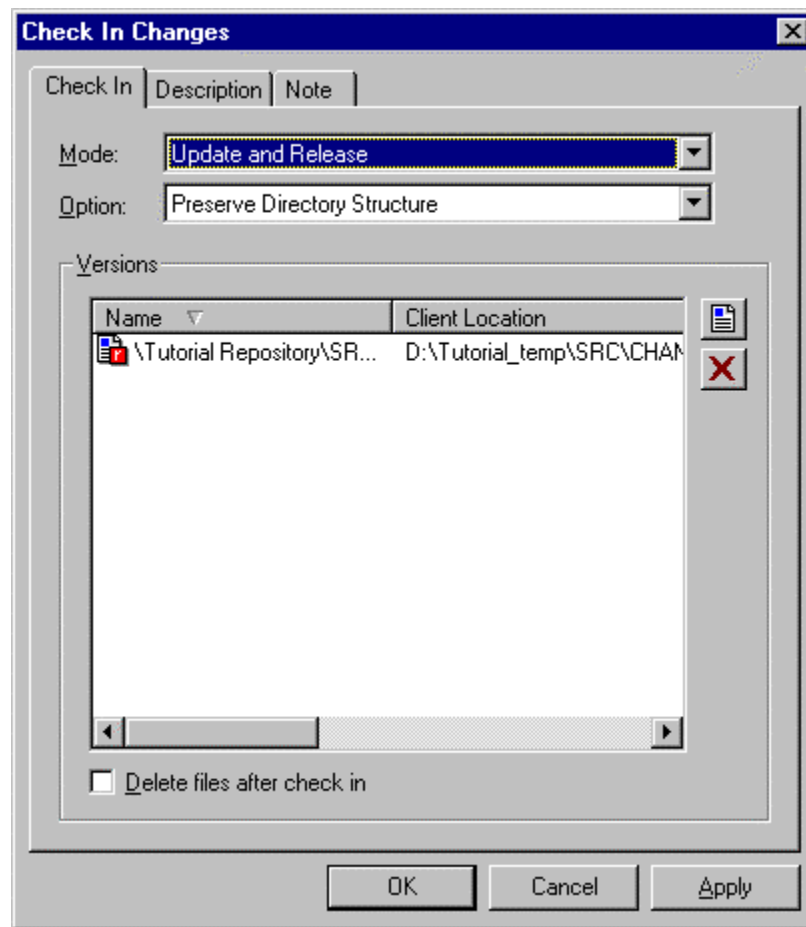
1. Find the file CHANGESI.C on your local filesystem and open it in any text editor.
2. Change line 9 to read "Only 10..." instead of "Only 9..."
3. Save your changes and close the file.

## Step 4: Check In the Change

Bob has fixed the problem and he needs to check the new version of the file into Harvest.

1. If you exited Harvest earlier, log in to the workbench as again now as BobS.
2. Click OK on the Default Context dialog.
3. Right-click the package TMR#1001 and choose Processes, Check In Changes from the shortcut menu. Notice that the file shows up in the Files list box because it is associated with the package TMR#1001.
4. Choose Update and Release from the Mode drop-down list. The Update and Release option creates a new version on the trunk and removes the lock from the version so it can be checked out again later.
5. Choose Preserve Directory Structure from the Option drop-down list. Because you checked out the file with the Preserve and Create Directory Structure option, the work area directory structure matches the repository structure. Because you do not want this modified item to be stored in a different repository directory than the original item, it is important to use this option.

Your Check In dialog should now look similar to the following:

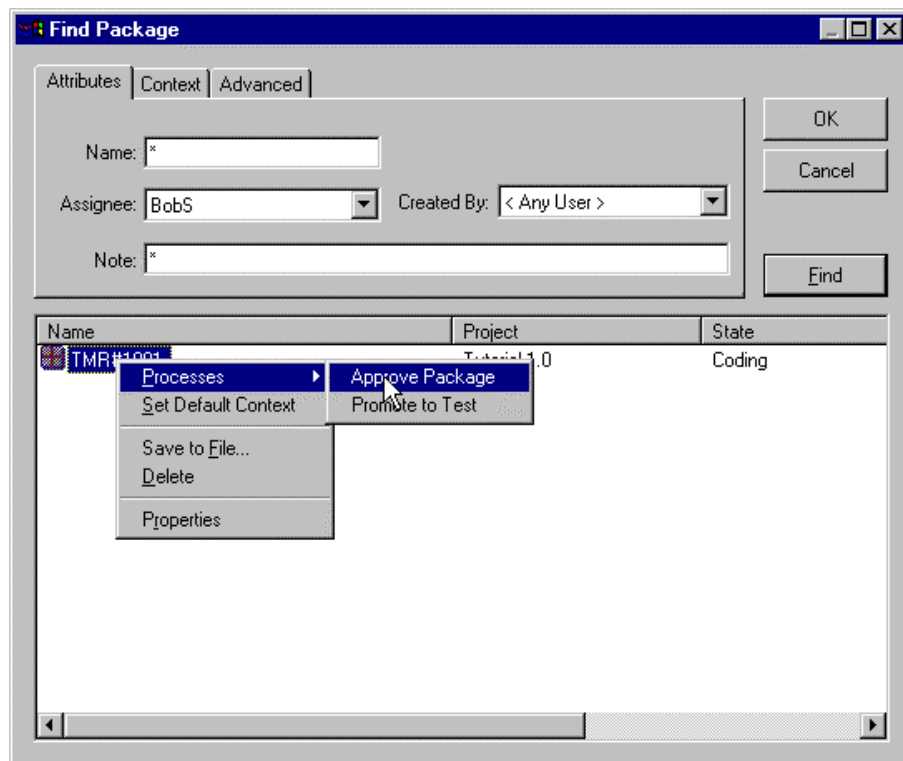


6. Click the OK button to execute the check in.
7. Notice in the output log that a new version of the file has been created.
8. Expand the Tutorial 1.0 project, then the Coding state, and then the Data Views folder.
9. Navigate the Data Views folder to the SRC directory.
10. Double-click on CHANGESI.C. Notice that there are now two versions of CHANGESI.C listed: the base version and version 1.
11. To view your change, right-click CHANGESI.C version 1. Select View Version to open a Harvest document showing the contents of CHANGESI.C. Verify the change you made. Click on the "x" in the upper right-hand corner to close the document. BobS exits the Harvest Workbench.

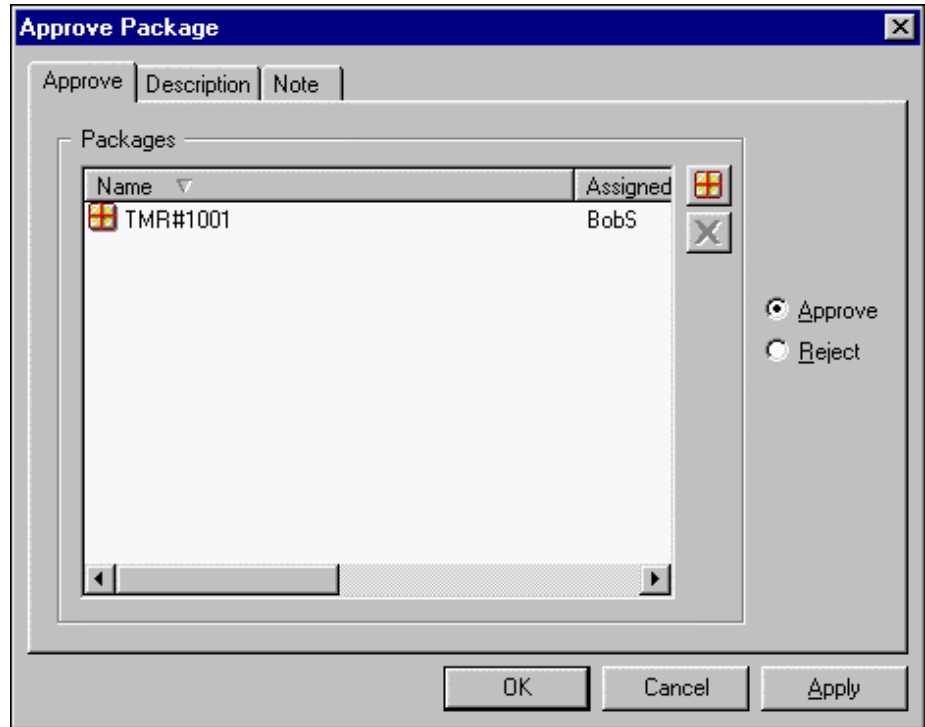
## Step 5: Approve the Package

Bob is done with his changes, but Jim needs to approve the package before it can be promoted to the Test state. Exit Harvest by choosing File, Exit from the main window menu.

1. Log in to the workbench as JimE.
2. On the Projects tab, click the Find Package icon (red package) on the toolbar to open the Find Package dialog.
3. Leave the defaults on all fields except from the drop-down list of the Assignee field choose BobS. Click the Find button.



- On the packages list of the Find Package dialog, right-click the package TMR#1001 and choose Processes, Approve Package from the shortcut menu. The following dialog will appear:



- You can see that BobS has only one package assigned to him. Select the Approve button and then click OK to approve the package. (If more packages were listed, you could have selected only the ones you wanted to approve and then click the OK button.)
- On the Find Package dialog, click OK to close it.

## Step 6: Promote the Package to Test

The package has been approved and Jim can promote it to the Test state so that he can start testing the changes.

- Locate the package using the Find Package dialog.

- or

On the Projects tab, expand the Tutorial 1.0 project, then the Coding state and then the Packages folder.

- Right-click the package TMR#1001 and choose Processes, Promote to Test from the shortcut menu.

3. On the Promote to Test dialog, visually confirm the package you want to promote and click OK.
4. Expand the Test state and then the Packages folder. Notice that TMR#1001 is now listed there instead of the Coding state.
5. Exit Harvest by choosing File, Exit from the main window menu.

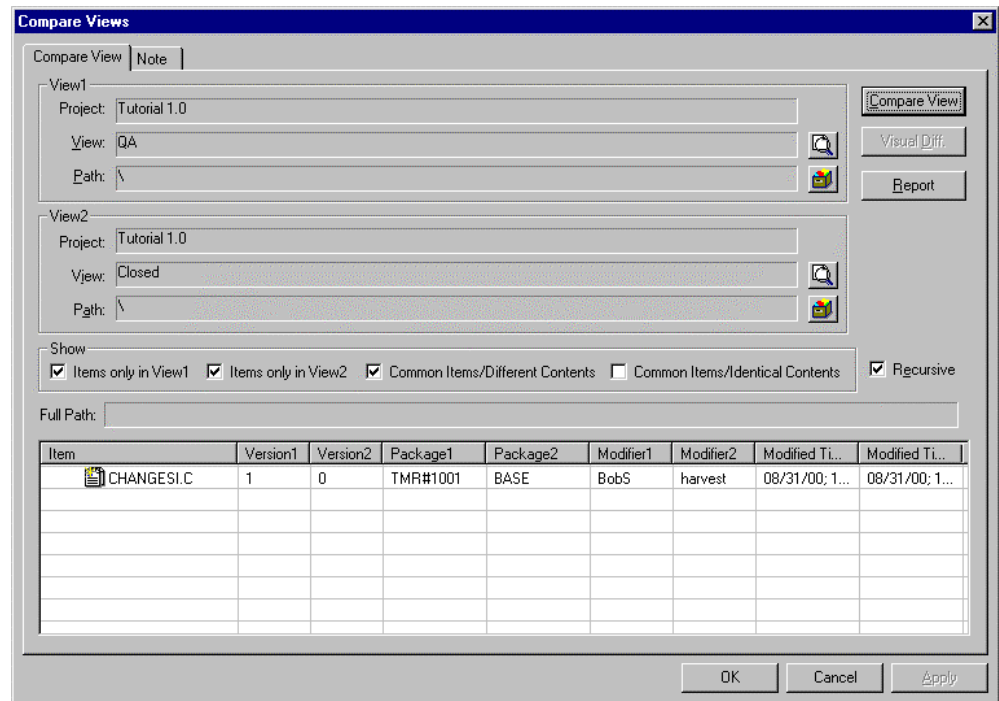
## Step 7: Execute a Compare Views

Rick wants to see what the differences are between the versions in the QA view and the versions in the Closed view. To do this, he can execute the compare views process.

1. Log in to the workbench as RickC.
2. On the Default Context dialog, choose Test on the State drop-down list. Click OK.
3. Right-click the Test state and choose Processes, Compare Views from the shortcut menu.
4. The View1 area should already list the Tutorial 1.0 project and the QA view. If it does not, click on the buttons in the View1 area to open dialogs in which you locate and select the QA view and the Tutorial Repository.
5. The View2 area needs to have the Tutorial 1.0 project and the Closed view listed. Click the buttons next to the to View area and then select the correct view.
6. The Show area offers you various options to select for your view comparison. Select the Items only In View1, Items only in View2 and Common Items/Different Contents boxes. Deselect Common Items/Identical Contents. This selection will show only the items that differ between View1 and View2.



7. Select Recursive to show all subdirectories.

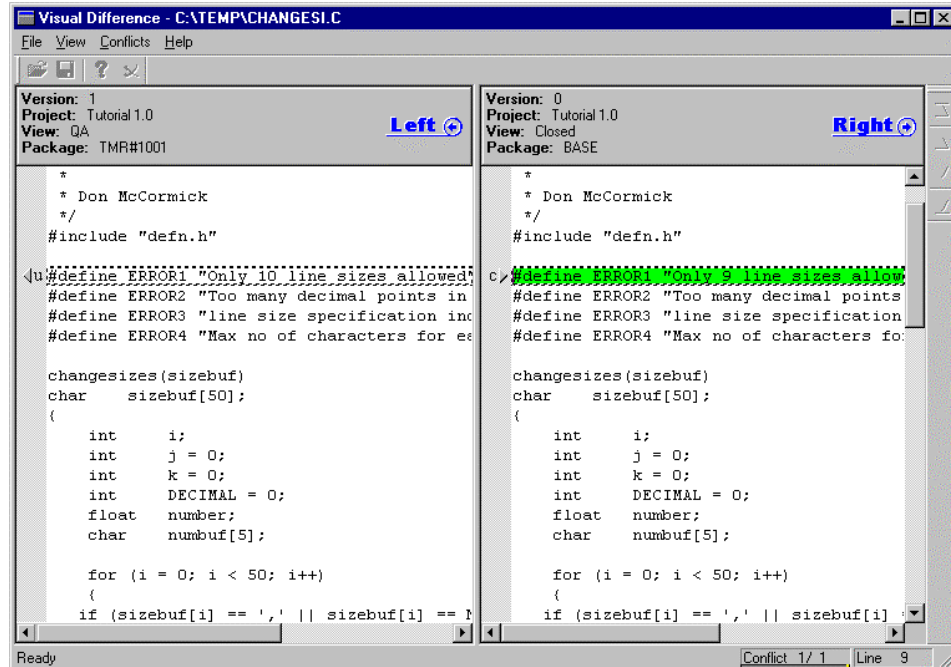


8. Click the Compare View button.
9. A list of items that are common to both views but that differ in content are displayed in the list box. Notice CHANGESI.C is listed and its two versions described. If you had also wanted to see items that were common to both views and identical in content, you would have additionally selected Identical Common Items before clicking the Compare View button.
10. To generate a report, click the Report button. This sends a report of the listing to the output log from which you can save it or print it.
11. On the Compare View dialog, you are notified, "Report is sent to the output tab successfully." Click OK. Look at the output log to see the report.

## Step 8: Perform a Visual Difference

Rick wants to have a detailed look at the differences between the two versions of CHANGESI.C. To do this, the Visual Difference window is invoked by clicking the Visual Difference button on the Compare Views process execution dialog. The Visual Difference window displays detailed, line-by-line differences between two files. When the dialog is invoked, the versions being compared are displayed with common blocks (lines the same in both versions) and conflict blocks. Because the Compare Views process is read-only, changes cannot be made to the files.

1. In the Compare Views dialog listing, select the CHANGESI.C row.
2. The Visual Diff. Button becomes enabled; click on it.



The Visual Difference window opens. It has two panes: the Left pane and the Right pane. The title bar displays the version names and their contexts. The Left pane contains the current project and view context when the Compare Views dialog was invoked. The Right pane contains the comparison view that was selected by the compare views process.

Text characters beside lines and color codes indicate the status of the text. On the left pane, note the line marked with a "u" indicating unchanged text. On the right pane, note the green line marked with a "c" indicating changed text. The other possible status indicators are:

- a indicating added text. The color code is yellow.
- d indicating deleted text. The color code is gray.

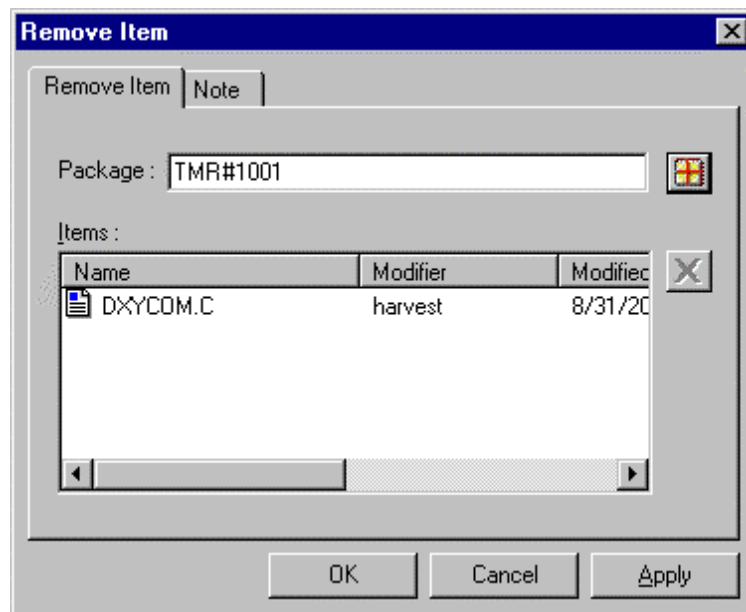
The Conflict meter at the bottom of the window shows your current conflict location and the total number of conflicts. The line number of the current conflict is also displayed in the status bar.

3. Using the vertical scroll bar, browse the files. You can see that the panes are synchronized to ensure that the panes are displaying the same conflict lines. Notice the green text line; this indicates the changed line. (The horizontal scrollbars do not synchronize the panes.)
4. Close the Visual Difference window and then the Compare Views dialog.

## Step 9: Remove an Item

Rick found an item in the repository that is no longer needed and should be deleted. Harvest does not allow users without administration privileges to delete items from the repository. Rick needs to execute the remove item process instead. The remove item process logically deletes an item from view so that it does not appear in selection lists and cannot be acted upon.

1. On the Projects tab, expand the Tutorial 1.0 project, the Test state, and then the Data Views folder.
2. Navigate to the SRC directory and double-click on it.
3. Right-click DXYCOM.C and choose Processes, Remove Item from the shortcut menu.
4. On the Remove Item dialog, click the package icon and choose TMR#1001 from the list.
5. The DXYCOM.C item should already be listed in the Items field because you selected it on the workbench. Click OK to remove the item.

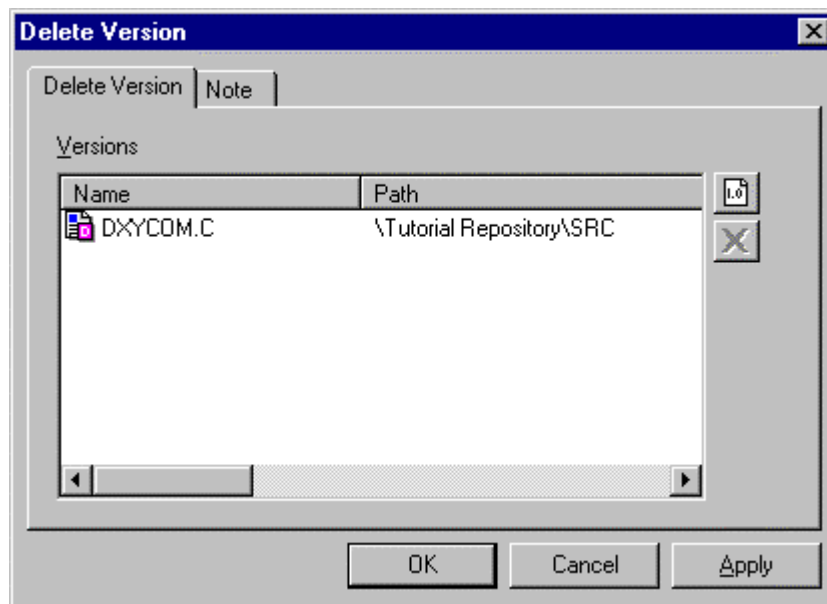


6. Expand the packages folder in the Test state, and then expand the package TMR#1001.
7. Double-click on the Versions folder located beneath the package TMR#1001.
8. In the list view, notice that DXYCOM.C is not actually deleted, but a new version is created with a special "D" icon. The "D" version indicates that the item has been removed from view.

## Step 10: Delete a Version

Now that Rick removed DXYCOM.C, he realizes that he actually needs the item and should not have removed it. To restore the item, all he needs to do is delete the "D" version.

1. On the Projects tab, expand the packages folder in the Test state, and then expand the package TMR#1001.
2. Double-click on the Versions folder located beneath the package TMR#1001.
3. Right-click the "D" version of DXYCOM.C and choose Processes, Delete Version from the shortcut menu.



4. On the Delete Version dialog, click OK to delete the selected version.

The delete version process can also be used to physically delete actual versions of items. Once a version is deleted, it cannot be retrieved, so this process should be used very carefully.

## Step 11: Demote the Package

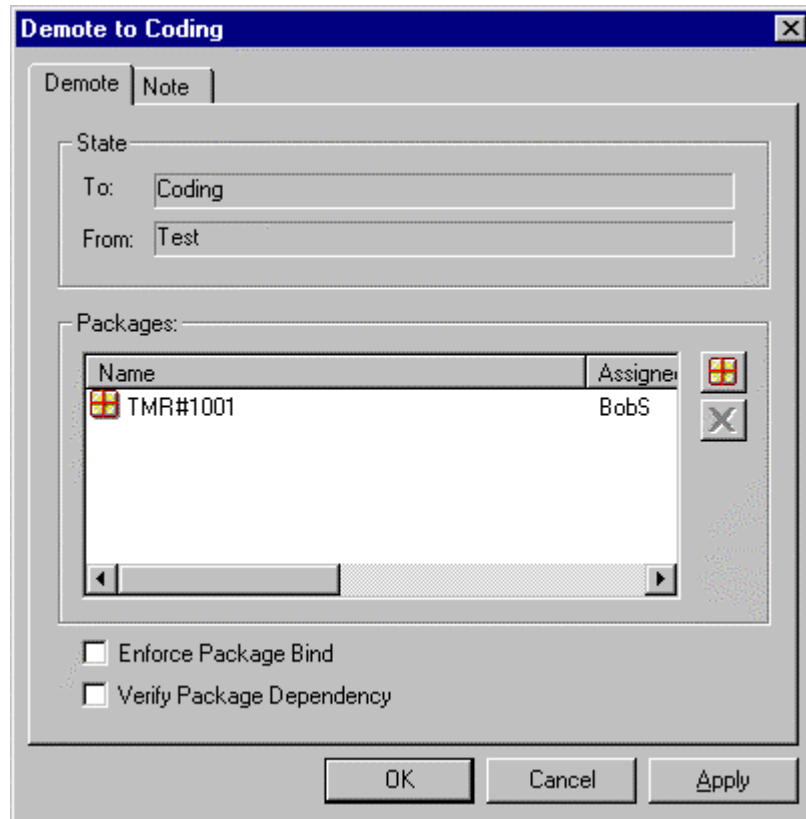
Rick has determined that the changes made by Bob did not fix the problem. He now needs to demote the package to the Coding state so that Bob can fix it.

1. Locate the package using the Find Package dialog.

- or

On the Projects tab, expand the Tutorial 1.0 project, then the Test state, and then the Packages folder.

2. Right-click the package TMR#1001 and choose Processes, Demote to Coding from the shortcut menu.



3. Confirm the package you want to demote and execute the process by clicking OK.
4. Expand the Coding state, and then the Packages folder. Notice that TMR#1001 is now listed there instead of the Test state.

In addition, if you expand package TMR#1001, and then select Versions, you will see the version of CHANGESI.C associated with that package.

## In Review

In this chapter, you learned the difference between the administrative aspects of Harvest and the everyday operational aspects. You also learned how to perform the most common functions of Harvest, including:

1. Creating a change package and editing the associated form.
2. Promoting a change package.
3. Checking out an item for update.
4. Checking in a changed file.
5. Approving a change package.
6. Viewing the differences between items in two different views.
7. Viewing the differences between two versions.
8. Removing an item.
9. Deleting a version.
10. Demoting a package.

# Advanced Exercises

---

This chapter covers two advanced, but commonly used, Harvest functions: concurrent development and snapshots.

## Instructions for This Chapter

You should follow the exercises in this chapter while using your client machine. You must have completed Chapter 3 successfully before beginning this chapter. For best results, complete Chapter 4 before you begin this chapter.

You will log in and out of Harvest using some of the Harvest user names you created in Chapter 3. If your client is running a Microsoft Windows environment, you will simply log out of Harvest each time you change users.

## Concurrent Development

Before you begin this exercise, you need to understand how concurrent development works.

When the Harvest administrator sets up the check out process, the check out modes needed by the end-users are selected. The Update mode allows only one person to check out an item at a time. Harvest does not allow anyone to check out that same item for update until it has been checked back in or the lock has been removed. The Concurrent Update mode of check out allows multiple copies of an item to be checked out and updated at the same time.

Harvest stores deltas (changes) two different ways, depending on the mode of check out used. When the Update mode is selected, the new version created during the subsequent check in process is placed on what is called the main line, or trunk. If the Concurrent Update mode of check out is used, when the item is subsequently checked back in, Harvest creates what is called a branch. For each different package that is used to check out an item for concurrent update and check it back in, a separate branch is created.

When versions are stored on a branch, they are not available to be promoted to the next state. To make these versions available, they must be merged back onto the trunk. This is accomplished using the Concurrent Merge process.

The Concurrent Merge process looks to see if any versions greater than the one the branch is based on exist on the trunk. If no newer versions exist, the process moves the branch version to the trunk, creating a new version. If a newer version of the same item already exists on the trunk, the item will be given a merge tag. When an item is merge-tagged, you must use the Interactive Merge process to resolve the conflicts between your version and the latest version on the trunk.

## Step 1: Create Two Packages

1. Log in to the workbench as JimE.
2. Choose the Assign state on the State drop-down list of the Default Contest dialog. Click OK.
3. Right-click the Assign state and choose Processes, Create Package from the shortcut menu. This will open the Create Package process execution dialog.
4. Change the contents of the Name field to **TMR#1002** and click OK.
5. Repeat steps 3 and 4, changing the contents of the Name field to **TMR#1003**.
6. Promote both packages to the Coding state. See Chapter 4, Step 6: Promote the Package to Test if you need help.

You now have two new packages ready for the exercise. Remember, in order to check out an item version for update, you must have a package to associate with it.

Jim is now finished with his development manager duties. Go ahead and exit Harvest by selecting File, Exit from the main window.

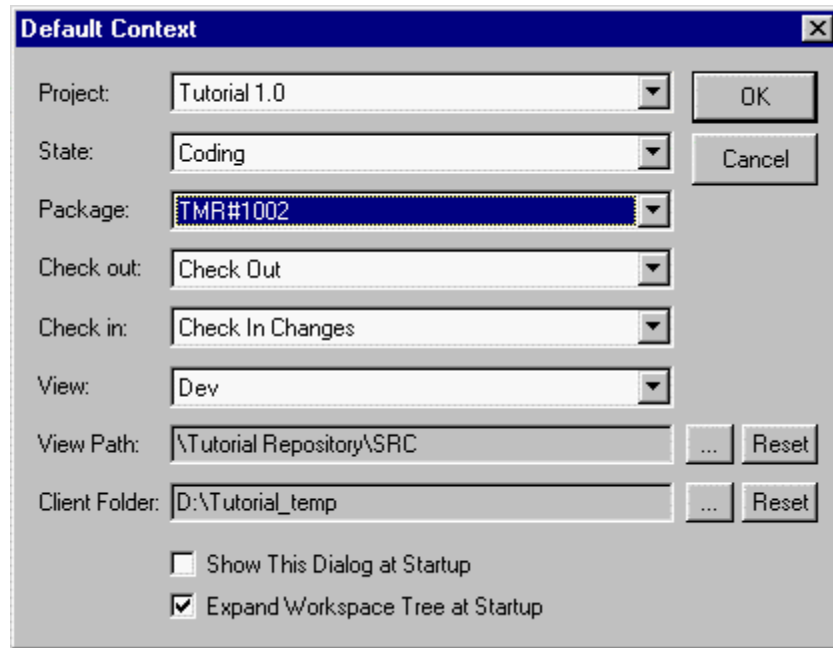
## Step 2: Check Out the File for Concurrent Update

Mary needs to check out the source code file GETOPT.C to make some changes. Using Concurrent Update mode will allow Bob to check out the same version later.

1. Log in to the workbench as MaryT.
2. On the Default Context dialog, use the drop-down lists to choose the Tutorial 1.0 project, Coding state, TMR#1002 package, check out and check in processes, and Dev view.

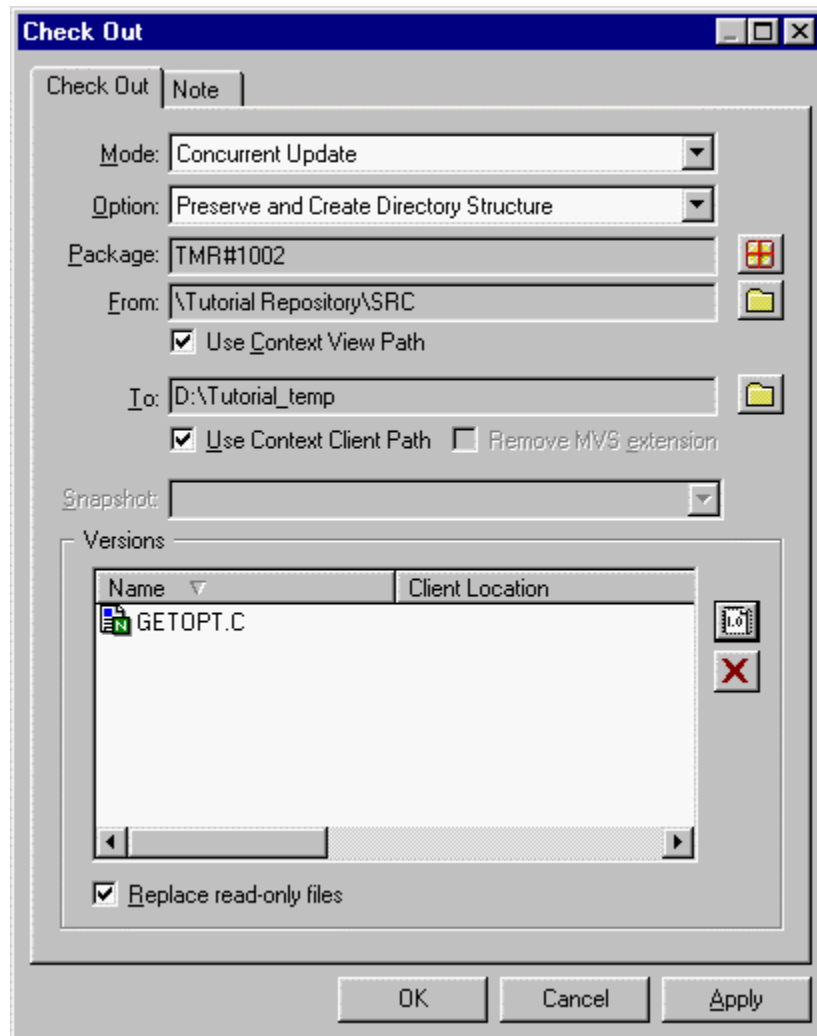
In order for Harvest to synchronize the location of the file after you have checked it out to your filesystem and the location of the item within the repository, you must specify the View Path and Client Folder locations. Click the buttons next to the path fields to browse for and set the view path and your client folder. Click OK.





3. Right-click TMR#1002 and choose Processes, Check Out from the shortcut menu.
4. Choose Concurrent Update from the Mode drop-down list.
5. Choose Preserve and Create Directory Structure from the Options drop-down list.
6. Click the Select Version icon above the red X to open the Select Version dialog to locate and select the file GETOPT.C. Check the Recursive box to enable a recursive search. Click OK to return the file to the Check Out dialog. If you need to review using the Select Version dialog, see Step 3: Check Out an Item For Update, in Chapter 4.
7. Ensure that the package TMR#1002 is shown in the Package field.
8. The To field should show the working directory you specified in the Default Context dialog. If it does not, click the button to the right of the field, then navigate your client file system until you locate your working directory. After you have it selected, click OK.
9. Click the Replace Read-only Files check box to select it.

10. Your Check Out dialog should now look similar to the following:

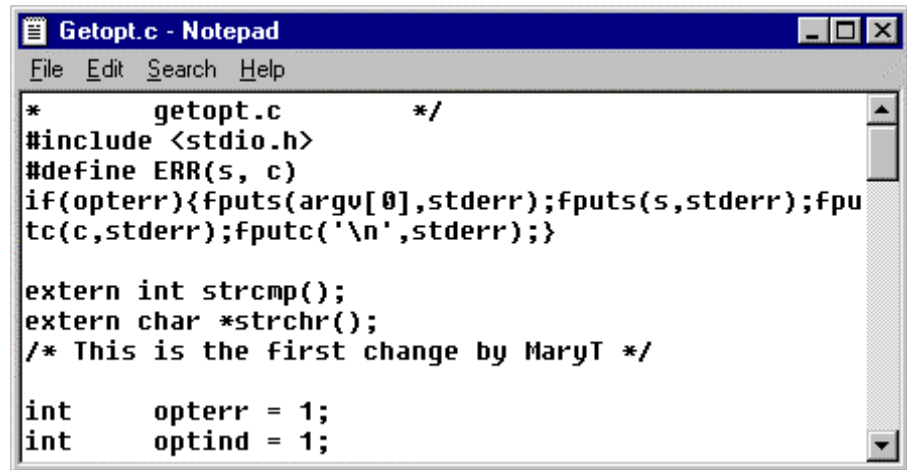


11. Click the OK button to execute the check out.

12. Check the output log to make sure the file was checked out successfully.

### Step 3: Edit the File

1. Using a text editor, open the file GETOPT.C.
2. Delete the slash (/), which is the first character on the first line of the file, and add a comment line to the program, as shown below.



```
*      getopt.c      */
#include <stdio.h>
#define ERR(s, c)
if(opterr){fputs(argv[0],stderr);fputs(s,stderr);fpu
tc(c,stderr);putc('\n',stderr);}

extern int strcmp();
extern char *strchr();
/* This is the first change by MaryT */

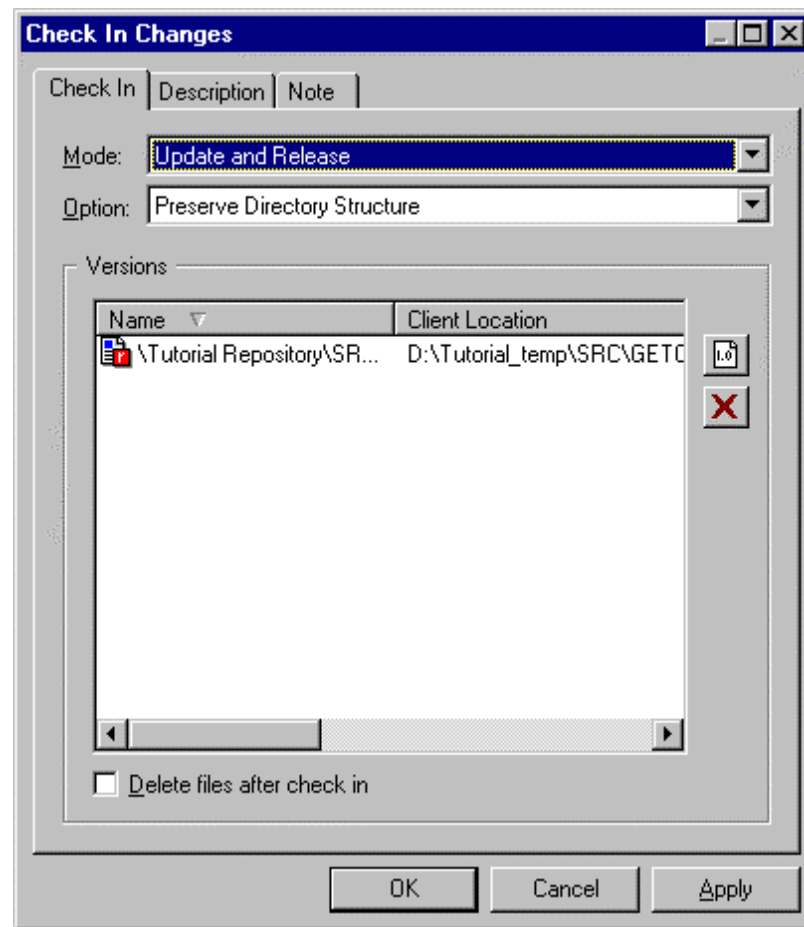
int      opterr = 1;
int      optind = 1;
```

3. Save the file and return to Harvest.

### Step 4: Check In the File

1. You must be logged in to the workbench as MaryT, if not log in as MaryT.
2. On the Projects tab, navigate to the Tutorial 1.0 project and the Coding state.
3. Navigate to the package TMR#1002 and right-click it.
4. Choose Processes, Check In Changes from the shortcut menu to open the Check In Changes process execution dialog.
5. The Versions list box should show the GETOPT.C version you checked out and edited.
6. Ensure that Update and Release is selected on the Mode drop-down list.
7. Ensure that Preserve Directory Structure is selected on the Options drop-down list.

Your Check In dialog should now look similar to the following:



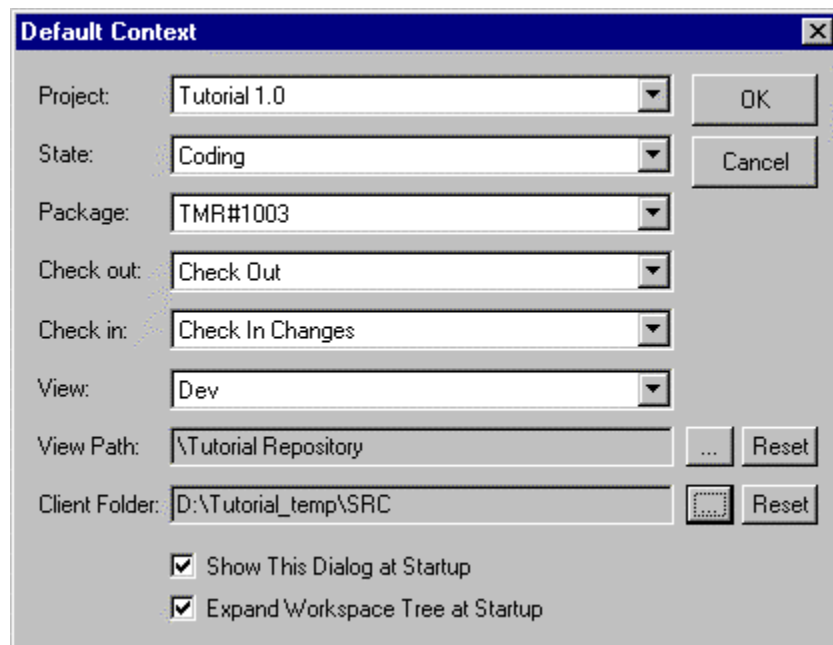
8. Click the OK button to execute the check in.

Notice in the log that the version number created by the check in process is 0.1.1. The 0 identifies the version of GETOPT.C on the trunk that Mary originally checked out. The first 1 in the number indicates that it is the first branch from version 0. The second 1 indicates the version number on branch 1. This three-part version number shows you that the version exists on a branch, not on the trunk.

9. Exit Harvest by choosing File, Exit from the main window menu.

## Step 5: Check Out for Concurrent Update Again

1. Log in to the workbench as BobS.
2. On the Default Context dialog, use the drop-down lists to choose the Tutorial 1.0 project, Coding state, TMR#1003 package, check out and check in processes, and Dev view.
3. In order for Harvest to synchronize the location of the file after you have checked it out to your filesystem and the location of the item within the repository, you must specify the View Path and Client Folder locations. Click the buttons next to the path fields to browse for and set the view path and your client folder. Click OK.



6. Expand the Data Views folder and the Dev View folder.
7. Expand the Tutorial Repository and then the /SRC directory. Click on the /SRC directory; a list of its contents will appear in the list view.
8. Double-click on GETOPT.C. This will bring up a list of versions for this item. Right-click GETOPT.C version 0 and choose Processes, Check Out from the shortcut menu. Notice that GETOPT.C version 0 is already displayed in the Versions list box of the Check Out process execution dialog. This is because you selected it from the version list before invoking the Check Out process, so Harvest assumes that is the version you want to check out.
9. Choose Concurrent Update from the Mode drop-down list.
10. Choose Preserve and Create Directory Structure from the Options drop-down list.
11. Ensure that the package TMR#1003 is shown in the Package field.

12. The To field should show the working directory you specified in the Default Context dialog. If it does not, click the button to the right of the field, and then navigate your client file system until you locate your working directory. After you have it selected, click OK.
13. Click the Replace Read-only files check box to select it. Click OK.
14. Check the log to make sure the file was checked out successfully.

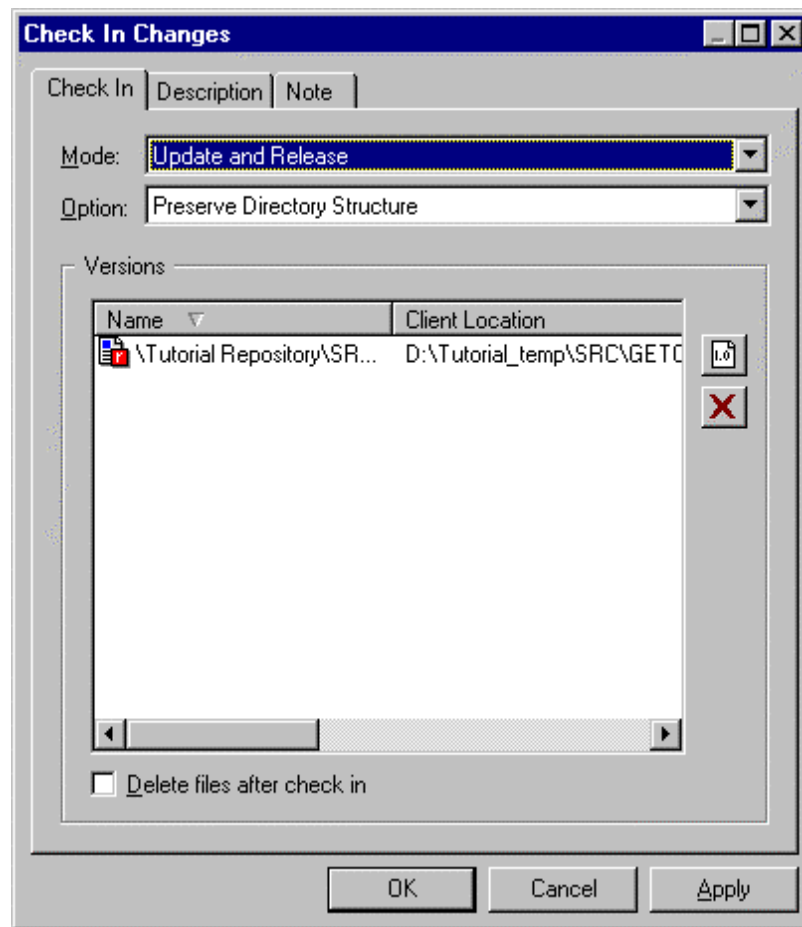
## Step 6: Edit the File

1. Navigate to your working directory and then to the /SRC directory within it.
2. Using a text editor, open the file GETOPT.C.
3. Insert the following line at the beginning of the file:  
`/* this is Bob's first change */`
4. Insert the following line after the line that reads int:  
`/* this is Bob's second change */`
5. Save the file and return to Harvest. If you must log to the workbench in again, do so as BobS.

## Step 7: Check In the File

1. On the Projects tab, navigate to the Tutorial 1.0 project, the Coding state and the package TMR#1003.
2. Right-click the package TMR#1003 and choose Processes, Check In Changes from the shortcut menu.
3. Choose Update and Release from the Mode drop-down list.

- Choose Preserve Directory Structure from the Options drop-down list.



- Click the OK button to execute the check in.
- Notice in the output log that the version number created by the check in process is 0.2.1. The 0 identifies the version of GETOPT.C on the trunk that Bob originally checked out. The number 2 indicates that it is the second branch from version 0. The number 1 indicates the version number on branch 2.

## Review the Current Status

- On the Projects tab, navigate to the Tutorial 1.0 project and the Coding state.
- Expand the Data Views folder and the Dev View folder.
- Expand the Tutorial Repository and then the /SRC directory.

4. Double-click on GETOPT.C. In the list view, notice the three versions of GETOPT.C that exist. Next to each version, the package that was used to check it in is displayed, along with the person who checked it in and when.

Version 0, the base version, is the one that was originally created in Chapter 3 during the load repository exercise, and versions 0.1.1 and 0.2.1 were just created by MaryT and BobS.

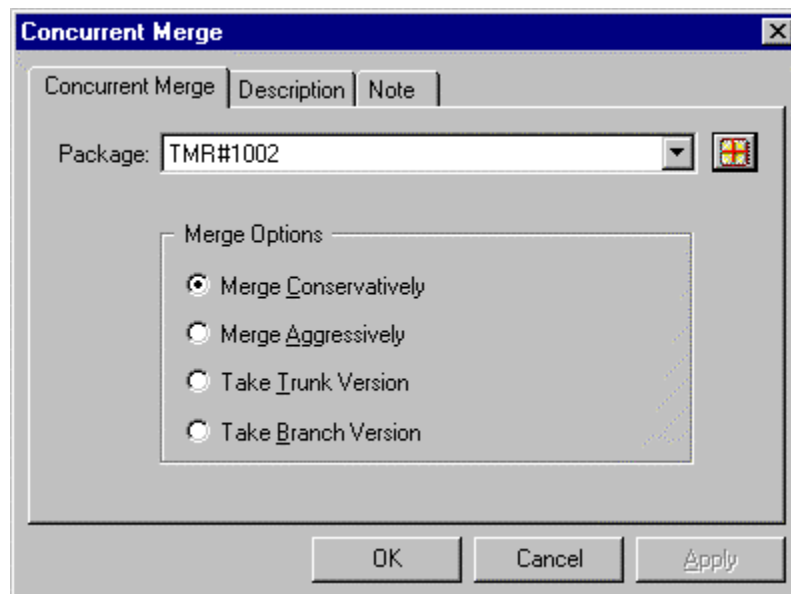
5. Go ahead and exit Harvest by choosing File, Exit from the main window menu.

## Step 8: Concurrent Merge

Mary is finished with TMR#1002 and is ready to promote it to the Test state, but the version associated with the package is still on a branch. She will need to perform a concurrent merge to merge her branch version into the trunk.

1. Log in to the workbench as user MaryT.
2. On the Default Context dialog, use the drop-down list to choose The Tutorial 1.0 project, Coding state and TMR#1002 package. Click OK.
3. Right-click TMR#1002 and choose Processes, Concurrent Merge from the shortcut menu. Notice that the merge is performed based on a package, rather than a single item. The merge process affects all items modified by the package and can merge multiple items simultaneously. In this case however, only one item is involved.

The following dialog will appear:





4. Click the OK button to perform the merge.

Notice in the log output that a normal version was created. This is because Mary's branch version was the first branch version merged into the trunk and does not conflict with version 0 on the trunk.






5. Exit Harvest by choosing File, Exit from the main window menu.

Bob is finished with TMR#1003 and is ready to promote it to the Test state, but the version associated with the package is still on a branch. He will need to perform a concurrent merge to merge his branch version into the trunk.

1. Log in to the workbench as user BobS.
2. On the Default Context dialog, use the drop-down list to choose The Tutorial 1.0 project, Coding state, TMR#1003 package and Dev view. Click OK.
3. Right-click TMR#1003 and choose Processes, Concurrent Merge from the shortcut menu. Notice that the merge is performed based on a package, rather than a single item. The merge process affects all items modified by the package and can merge multiple items simultaneously. In this case however, only one item is involved.
4. The Concurrent Merge dialog will appear showing TMR#1003 in the Package field. Click the OK button to perform the merge.

Notice in the log output that a merged version was created. This is because Bob's branch version conflicts with Mary's new trunk version.

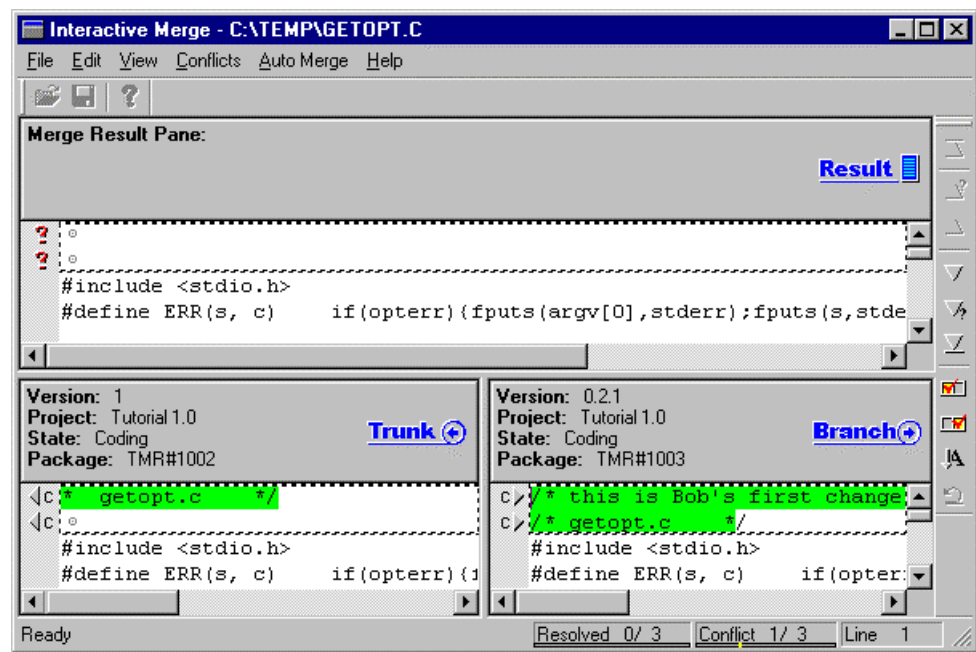
5. On the Projects tab, navigate to the Tutorial 1.0 project and the Coding state.
6. Expand the Data Views folder and the Dev View folder.
7. Expand the Tutorial Repository and then the /SRC directory.
8. Double-click on GETOPT.C. Notice that version 2 of GETOPT.C has a different icon next to it. This icon is a merge tag. Because Bob checked out version 0 of the item, and version 1 had been created before Bob checked his changes back in, Harvest realizes there is a conflict between version 1 and Bob's version 0.2.1, and thus designates Bob's new version 2 as merge-tagged.

Name	Path	Mapped v...	Statu
 GETOPT.C	\Tutorial Repository\SRC	0	N
 GETOPT.C	\Tutorial Repository\SRC	0.1.1	N
 GETOPT.C	\Tutorial Repository\SRC	0.2.1	N
 GETOPT.C	\Tutorial Repository\SRC	1	N
 GETOPT.C	\Tutorial Repository\SRC	2	M

In order to resolve the conflicts between the two versions, Bob must now perform an interactive merge. If no conflicts existed between the two versions, no merge tag would be created and the interactive merge process would be unnecessary.

## Step 9: Interactive Merge

1. On the Projects tab, navigate to the Tutorial 1.0 project and the Coding state.
2. Expand the Data Views folder and the Dev View folder.
3. Expand the Tutorial Repository and then click on the /SRC directory. Double-click on GETOPT.C.
4. In the list view, right-click GETOPT.C version 2 and choose Processes, Interactive Merge from the shortcut menu. The interactive merge process enables you to combine the changes made on unmerged branches with changes on the trunk or to resolve a merge-tagged version after it is created by the concurrent merge or cross-project merge process.



The Interactive Merge Process Execution dialog has three panes: the Merge Result pane, the Trunk pane and the Branch pane.

- The Merge Result pane is the top pane. The text in the Merge Result pane represents the final version of the file and will become the new version that is created when the file is checked in.
- The Trunk pane contains the trunk version of the file being merged.
- The Branch pane contains the branch version of the file being merged.

The two versions of the item being merged are displayed in the merge execution dialog with common blocks (lines identical in both versions) and conflict blocks. Conflict blocks are positioned side-by-side, and one block must be selected in favor of the other. The panes are synchronized to ensure that all three panes are displaying the same conflict lines. Text characters beside lines and color codes indicate the status of the text. Notice the green text lines, indicating conflicting text, and yellow text lines indicating added text.

The status bar at the bottom of the dialog contains two meters, Resolved and Conflict. The line number of the current conflict is also displayed in the status bar.

The Conflicts menu and a toolbar provide you with different options to navigate through the conflicts. The navigation options found in the Conflicts menu are the same as those found on the toolbar.

Options to select the trunk or branch conflicts are available from the Edit menu, shortcut menu, shortcut keys and the toolbar. Two options allow you to reset conflicts to an unresolved condition. Auto Merge options allow you to specify how the process will execute. During setup the administrator defined the options available.

There are two execution options on the dialog and in the File menu: Check In and Close.

- You cannot execute the interactive merge until all conflicts are resolved. The Check In option is disabled until all conflicts are resolved. Similarly, the Check In button on the dialog does not display until all conflicts are resolved. When all conflicts are resolved, choose Check In. Checking the file in will replace the merge-tagged version with a new, untagged version.
- If you do not want to merge the files, choose Close. If you have selected changes, a dialog displays in which you can confirm or cancel the Close. Close will disregard all your selections, and the file being merged will be returned to its original unmerged state.

To resolve the conflicts:

For the purpose of this tutorial, you want to accept the changes that Bob made and reject the version 1 changes.

5. Review the file, accepting all version 0.2.1 changes, which are displayed on the right-hand side. Doing this will eliminate the version 1 change. If you wanted to retain the version 1 change, you could accept that conflict block instead of the version 0.2.1 block.
6. From the menu, choose AutoMerge, Merge All Changes, Take Branch Conflicts. This will automatically select all changes and select the trunk conflict when a conflict between the trunk and branch occurs. Notice that the Merge Result Pane shows BobS' additional line, and the conflict meter shows all conflicts are resolved.

7. Click OK when you are finished. Notice on the output log that version 2 has been resolved and a normal version created. If you look at the versions in the /SRC directory, you will see the merge tag has been removed from version 2 of GETOPT.C.

## Step 10: Approve the Packages

MaryT and BobS are finished with their packages, they can promote them but Jim needs to approve the packages first.

1. Exit Harvest, then log in to the workbench as JimE.
2. On the Default Context dialog, choose the Tutorial 1.0 project, Coding state, and <No packages>.
3. On the workbench, click on the Packages folder in the Coding state. Select both packages TMR#1002 and TMR#1003 in the list view, right-click and choose Processes, Approve Package from the shortcut menu.
4. On the Approve dialog, confirm both packages you want to approve by clicking Approve (if it is not already selected) and then click the OK button to execute the approval.

**Note:** The output log says the Approval process was successful and the packages are now frozen. These packages can no longer be used for the check out or check in processes until they are promoted.

## Step 11: Promote the Packages

Rick is ready to start testing these packages, but JimE needs to promote them to the Test state first.

1. On the Default Context dialog, choose the Tutorial 1.0 project and Coding state.
2. On the workbench, click on the Packages folder. The list view will list the packages in the Coding state.
3. Select both packages TMR#1002 and TMR#1003, right-click and choose Promote to Test from the shortcut menu.
4. On the Promote dialog, confirm the packages you want to promote by clicking OK to execute the promotion.
5. Exit Harvest, and then log in to the workbench as RickC.

RickC tests the fix and finds that it passes quality assurance, so now he wants to promote the package to the Release state.

1. On the Default Context dialog, choose the Tutorial 1.0 project and the Test state.

2. On the Projects tab of the workbench, expand the Test state, and then the Packages folder.
3. Click on the Packages folder. The list view will list the packages in the Test state.
4. Select both packages TMR#1002 and TMR#1003, right-click and choose Promote to Release from the shortcut menu.
5. On the Promote dialog, visually confirm the package you want to perform and click OK to execute the promotion.

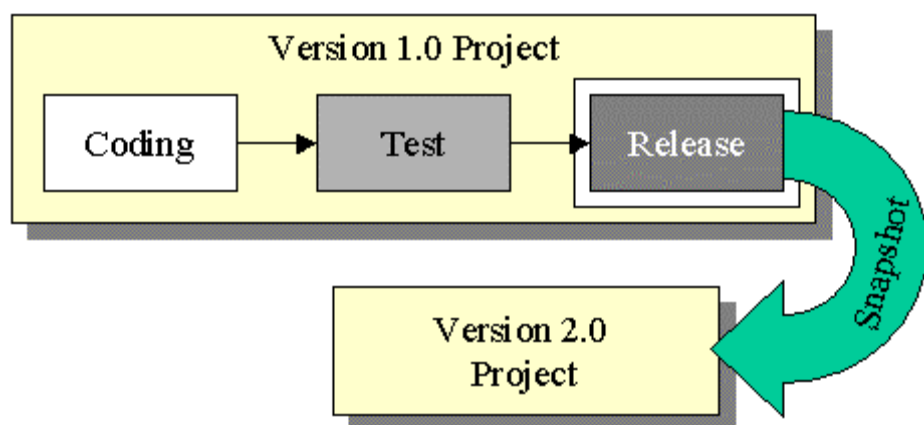
This completes the concurrent development exercise. RickC should exit Harvest now.

## Snapshots

Many software development organizations find that distributing software on a release-by-release basis is the most efficient means of maintaining the software. Here is how the release method works:

Version 1.0 of a product is completed and delivered to the customer. Maintenance activities continue on version 1.0, but work on a major new release, version 2.0, is scheduled to begin soon. The 1.0 maintenance team and the 2.0 project team will be working in parallel.

The 2.0 team would like to start their development based on the code that is in the Release state of the 1.0 project. When they start changing the code, they need their versions to be independent of the 1.0 project. The snapshot feature of Harvest allows this type of development scenario to happen.



A snapshot is an image of a working view at a specific point in time. Snapshots capture the latest versions of specified items and places them in a snapshot view. This snapshot view can be used by other projects as part or all the project's baseline. Snapshots can also be used to freeze a specific version of software for reference or investigation purposes.

In this exercise, you will learn how to create a snapshot view of the released code in the Tutorial 1.0 project. You will then create a new project for developing Version 2.0, using the 1.0 snapshot view for the baseline.

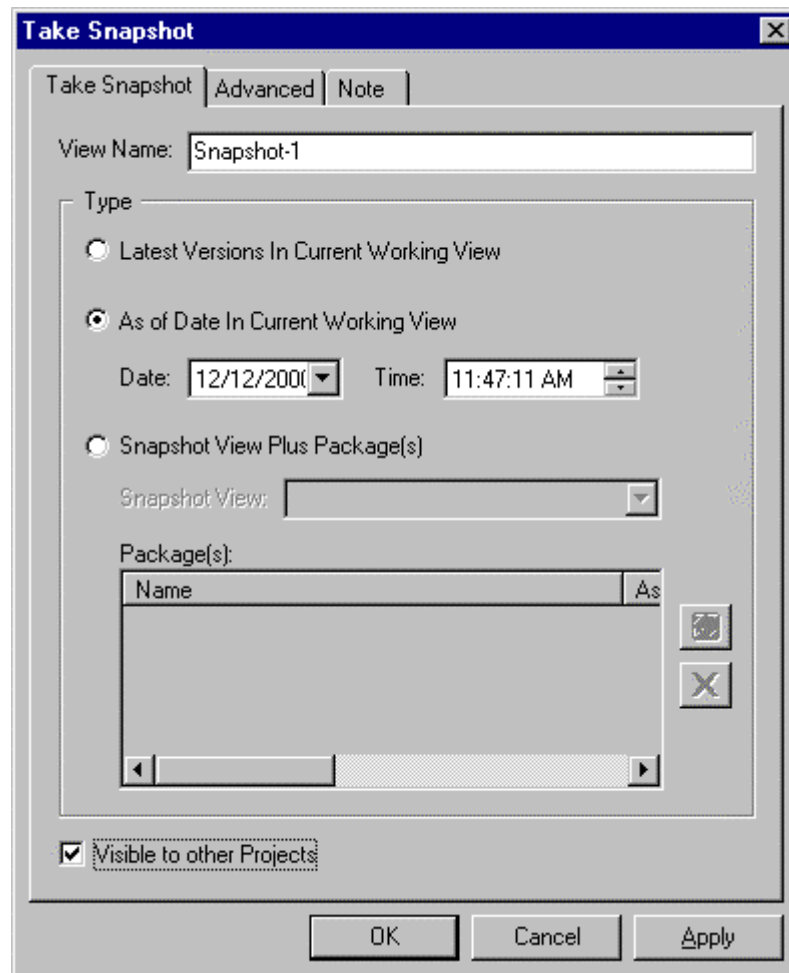
### Step 1: Create a Snapshot View

There are two methods for creating a snapshot view. One way is to create a new view in the Snapshot Views folder under the appropriate project. The other way is to execute the Take Snapshot process. First, you will create one using the Take Snapshot process.

#### The Take Snapshot Process

1. Log in to the workbench as JimE.
2. On the Default Context dialog, choose the Tutorial 1.0 project and the Release state.

3. Right-click the Release state and choose Take Snapshot from the shortcut menu. The following dialog will appear:



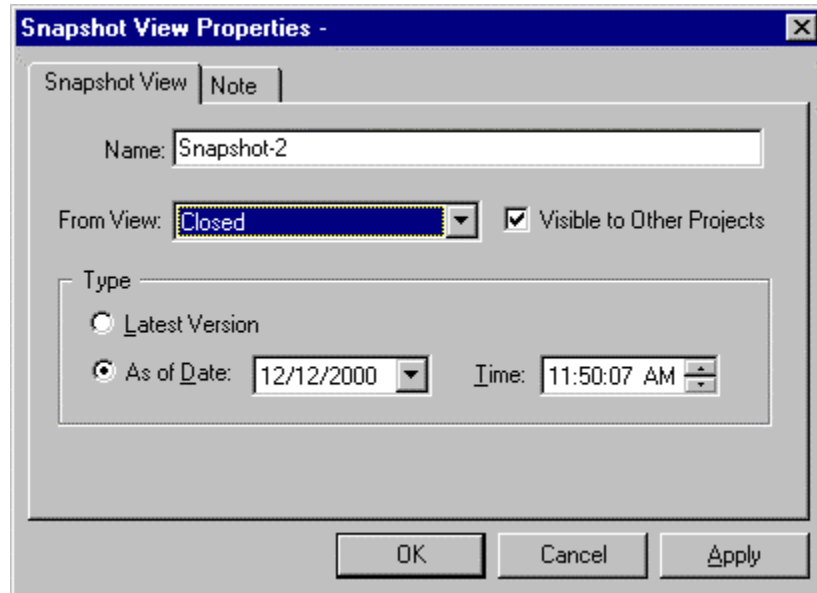
4. Enter Snapshot-1 in the View Name field.
5. Click on the As of Date in Current Working View button. The As of Date field should show the current date and time, and enable the user to change date and time fields if desired.
6. Click the check box next to Visible to Other Projects. This enables the snapshot view to be used by other projects.
7. Click OK.
8. Change your state context from Release to Snapshots and expand the Data Views under the Snapshots state. You will see Snapshot-1 listed.
9. Choose Snapshot-1 and then navigate down to the \Tutorial Repository\SRC directory.

10. Double-click on GETOPT.C. In the list view, you should see version 2 of GETOPT.C. This is because JimE promoted BobS' version 2 package to the Release state. In the final steps of this tutorial you will see that this version 2 from Tutorial 1.0 project's Snapshot-1 becomes the baseline version 0 for the new project Tutorial 2.0 and it contains all the changes made by BobS.
11. Exit Harvest by choosing File, Exit from the main window menu.

## The Snapshot View Method

Now you will create a snapshot view using the new snapshot view method.

1. Log in to the Administrator window as JimE.
2. Click the Lifecycles tab and expand the Active Projects folder.
3. Expand the Tutorial 1.0 project, and then expand the Data Views folder.
4. Right-click the Snapshot Views folder and choose New Snapshot View from the shortcut menu. The Snapshot View Properties dialog will appear.
5. In the Name field, type **Snapshot-2**.
6. Choose Closed from the From View drop-down list.
7. Click the check box next to Visible to Other Projects to enable it.
8. The As of Date field should show the current date and time.



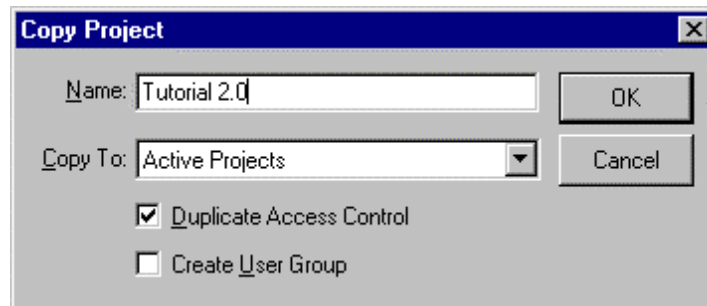
9. Click the OK button.
10. Click the Snapshot Views folder. You will see Snapshot-1 and Snapshot-2 listed.
11. Exit Harvest by choosing File, Exit from the main window menu.



## Step 2: Duplicate a Project

Now you need to create a new project for the Tutorial 2.0 release. You want to use the same kind of life cycle that you used in the 1.0 version, but you do not want to have to re-create all of the states, processes, and other objects that make up that life cycle. Fortunately, Harvest is designed to allow you to duplicate an existing project and use the new one as-is, or modify to meet other requirements.

1. If you are not still logged in as the tutorial user that was created in Chapter 2, do so now.
2. Click the Lifecycles tab and expand the Lifecycles Templates folder.
3. Right-click the Tutorial Template project, and choose Copy To from the shortcut menu. The Copy Project dialog will be displayed.
4. Enter Tutorial 2.0 in the Project Name field.
5. Choose Active Projects from the Copy To drop-down list and select the Duplicate Access Control check box. If you wanted a user group automatically created with the same name as the project, you could select the Create User Group check box. For this exercise, leave it unselected.

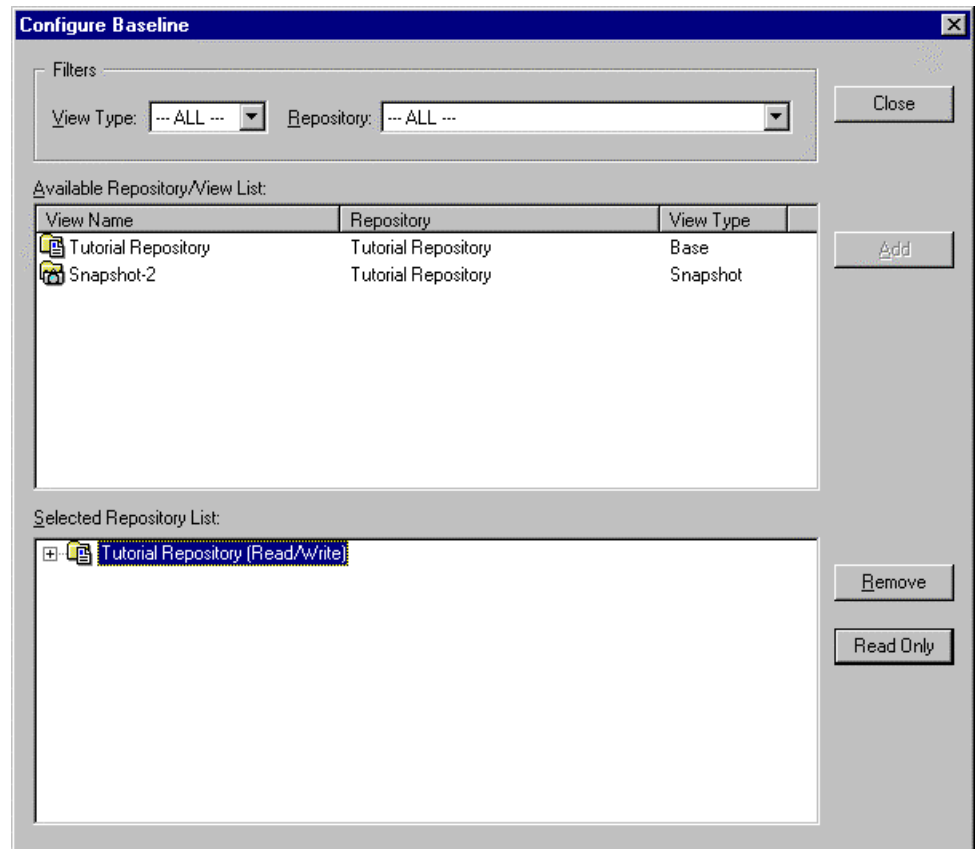


6. The Tutorial 2.0 project is created and is located in the Active Projects folder.

## Step 3: Establish the Baseline

You have created two different snapshot views, but you will only be using the one from the Release view to base your new project on.

1. Expand the Tutorial 2.0 project, and then expand the Data Views folder.
2. Right-click the Baseline folder and choose Configure Baseline from the shortcut menu. The Configure Baseline dialog will appear.



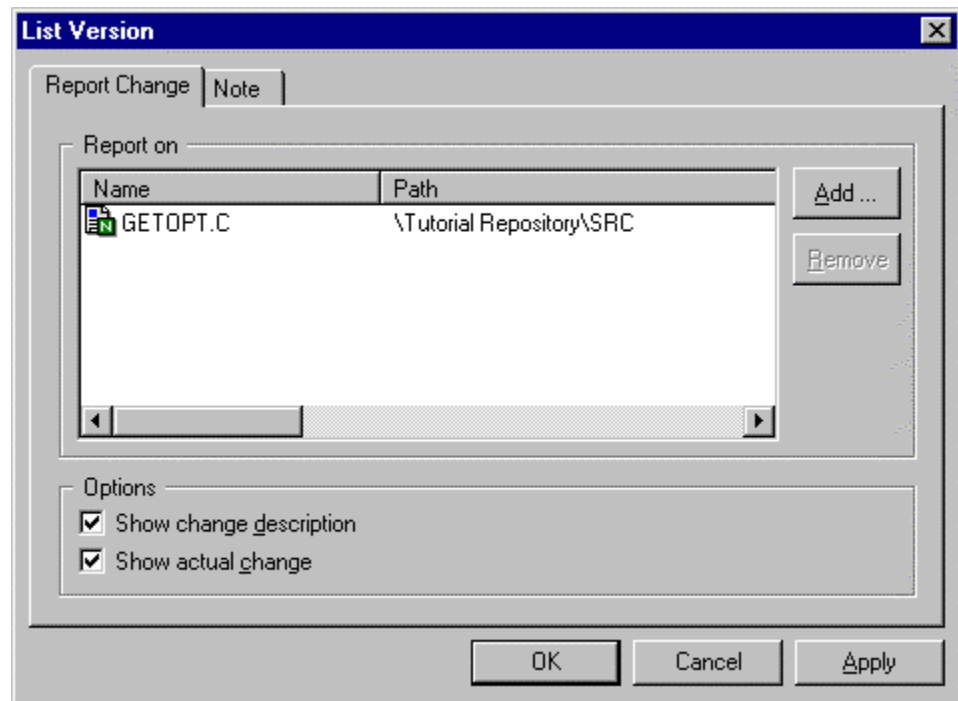
3. Select Snapshot-1 from the Available Repository/View List and then click the Add button.
4. Click Yes on the confirmation dialog.
5. You will see Tutorial Repository listed in the Selected Repository List. Select it and click the Read/Write button.
6. Click Yes on the confirmation dialog.
7. Click Close and then exit Harvest.

## Step 4: Run the List Versions Process

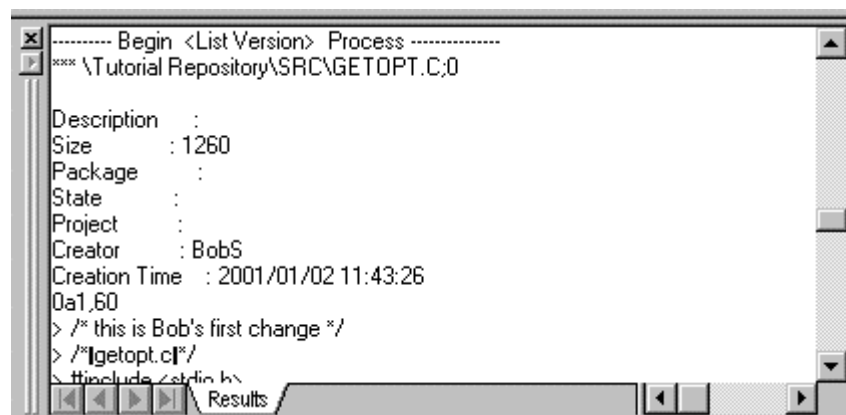
Now that you have created your new project and have established the baseline using Snapshot-1, you can check that the baseline contains the correct versions of the correct items by using the list versions process. The list versions process is a report that allows you to see the actual content of a particular version.

1. Log in to the workbench as JimE.
2. On the Default Context dialog, choose the Tutorial 2.0 project and the Coding state.

3. On the workbench, navigate to the Data Views folders and the \Tutorial Repository\SRC directory.
4. Double-click on GETOPT.C to list versions associated with this item. Right-click GETOPT.C and choose Processes, List Versions from the shortcut menu. The following dialog will appear.



5. Ensure that the Show actual changes and Show change descr check boxes are selected.
6. The version listed should already be \Tutorial Repository\SRC\GETOPT.C. If it is not, click the Add button next to the list box, then locate and select GETOPT.C and click OK.
7. Click the OK button. The following will appear in the log:



---

As you review the code, you will find all the changes that were included in Bob's version.

This completes the snapshot exercise. Exit Harvest by choosing File, Exit from the main window menu.